

NAVAL POSTGRADUATE SCHOOL
Monterey, California

19950414 126



DTIC
ELECTE
APR 18 1995
S C D

THESIS

**COMPUTER INTERFACE DEVELOPMENT FOR
THE PETITE AMATEUR NAVY SATELLITE
(PANSAT) SIMULATOR**

by

Thomas C. Calvert

December, 1994

Principal Advisor:
Second Reader:

Rudolf Panholzer
Tri Ha

Approved for public release; distribution is unlimited.

DTIC QUALITY INSPECTED 3

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE December, 1994	3. REPORT TYPE Master's Thesis		
4. TITLE AND SUBTITLE COMPUTER INTERFACE DEVELOPMENT FOR THE PETITE AMATEUR NAVY SATELLITE (PANSAT) SIMULATOR			5. FUNDING NUMBERS	
6. AUTHOR(S) Thomas C. Calvert				
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the US Government.				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) This thesis describes the development of personal computer interfaces for the Digital Control Subsystem (DCS) and Electrical Power System (EPS) of the Naval Postgraduate School's Petite Amateur Navy Satellite Simulator project. The work includes a brief description of related PANSAT and PANSAT simulator subsystems. The laboratory activity that provided the information for this work, namely, programming in the LabVIEW software programs G graphical programming language, involved the generation of programming code for controlling and reading data from the simulators component subsystems. Where appropriate, there is limited discussion of key satellite and simulator component designs and installations. Through the use of documentation and illustrative figures, the programming and display components of the interfaces are shown to be functional in the simulator environment. Instructions for making anticipated design accomodations are included.				
14. SUBJECT TERMS PANSAT, computer interface, graphical programming language, simulator			16. PRICE CODE	
			15. NUMBER OF PAGES 145	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	
NSN 7540-01-280-5500			Standard Form 298 (Rev. 2-89)	

Approved for public release; distribution is unlimited.

**COMPUTER INTERFACE DEVELOPMENT FOR THE PETITE AMATEUR
NAVY SATELLITE (PANSAT) SIMULATOR**

by

Thomas C. Calvert
Lieutenant, United States Navy
B.A., Weber State College, 1985

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

**NAVAL POSTGRADUATE SCHOOL
December, 1994**

Author:

Thomas C. Calvert

Thomas C. Calvert

Approved by:

Rudolf Panholzer

Rudolf Panholzer, Advisor

Tri T. Ha

Tri Ha, Second Reader

Michael A. Morgan

Michael A. Morgan, Chairman,
Department of Electrical and
Computer Engineering

Accession For	
NTIS	CRA&I <input checked="checked" type="checkbox"/>
DTIC	TAB <input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	

ABSTRACT

This thesis describes the development of personal computer interfaces for the Digital Control Subsystem (DCS) and Electrical Power System (EPS) of the Naval Postgraduate School's Petite Amateur Navy Satellite Simulator project. The work includes a brief description of related PANSAT and PANSAT simulator subsystems.

The laboratory activity that provided the information for this work, namely, programming in the LabVIEW software programs G graphical programming language, involved the generation of programming code for controlling and reading data from the simulators component subsystems. Where appropriate, there is limited discussion of key satellite and simulator component designs and installations.

Through the use of documentation and illustrative figures, the programming and display components of the interfaces are shown to be functional in the simulator environment. Instructions for making anticipated design accommodations are included.

TABLE OF CONTENTS

I. INTRODUCTION	1
II. THE PETITE AMATEUR NAVY SATELLITE (PANSAT)	3
A. OVERVIEW	3
B. THE DIGITAL CONTROL SUBSYSTEM (DCS)	5
C. THE ELECTRICAL POWER SUBSYSTEM (EPS)	10
III. THE PANSAT SIMULATOR	15
A. OVERVIEW	15
B. THE DIGITAL CONTROL SUBSYSTEM (DCS)	17
C. THE ELECTRICAL POWER SUBSYSTEM (EPS)	19
D. THE COMMUNICATIONS SUBSYSTEM (COMM)	19
IV. INTERFACE DEVELOPMENT	21
A. LABORATORY VIRTUAL INSTRUMENT ENGINEERING WORKBENCH (LABVIEW)	21
1. LabVIEW	21
2. Serial Port Communication	23
3. General Purpose Interface Bus (GPIB)	26
B. DCS INTERFACE	28
1. Ground Level Protocol	28
2. Low Level Commands	30
3. Control	32
4. High Level Commands	33
5. <i>TSWEEP2</i>	37
C. EPS INTERFACE	40
1. Power Supply Driver	40
2. Load Driver	40

3. Control Panels.....	43
V. CONCLUSIONS AND RECOMMENDATIONS.....	49
APPENDIX A. SIMULATOR DIGITAL CONTROL SUBSYSTEM (DCS)	
INTERFACE SOFTWARE CODE.....	51
APPENDIX B. SIMULATOR ELECTRICAL POWER SUBSYSTEM (EPS)	
INTERFACE SOFTWARE CODE.....	117
APPENDIX C. PERIPHERAL CONTROL BUS (PCB) ADDRESSES CURRENTLY	
DEFINED.....	131
LIST OF REFERENCES.....	133
INITIAL DISTRIBUTION LIST.....	135

I. INTRODUCTION

The process of designing, testing, and building a satellite includes the development of key individual components. Failure to complete a single component on time can disrupt the entire satellite development schedule. As with other complex systems, the ability to simulate the operation of a satellite in laboratory conditions can substantially enhance the performance of the deployed spacecraft. In addition to simulating the operation of the completed satellite system, a simulator can be designed to emulate the performance of key components. In this manner, it is possible to provide a stable testing platform with which to take both individual components and fully assembled subsystems from initial design to final implementation stages.

The Petite Amateur Navy Satellite (PANSAT) project at the Naval Postgraduate School in Monterey, California, will ultimately result in the deployment of a spacecraft in low-Earth orbit. An additional project underway involves the development of a PANSAT simulator. This simulator has been constructed using actual components, components designed specifically for the simulation, readily available standard components, and components that are functionally equivalent to actual components. In addition, a personal computer (PC) and a software program called LabVIEW have been used to create a capable and compatible simulator of PANSAT operations.

There are three main advantages to evaluation of components through simulation. First, evaluation of a completed component's performance within the total system is made possible by simulation of those components which are unfinished at the time of evaluation. Second, data on performance is easily monitored by the user at the computer terminal and can be presented in a format most suitable to the intended use. Third, the simulator is capable of simulating external factors that are encountered in the intended operating environment.

The purpose of this thesis is to describe the development of the simulator's Digital Control Subsystem (DCS) and Electrical Power Subsystem (EPS) interfaces. In the principal laboratory work for this thesis, these user interfaces were assembled using LabVIEW's graphical programming code. The result was the creation of an easy-to-use, graphical user interface for

controlling and monitoring the output from the PANSAT simulator. The DCS and EPS interfaces will be used as a basis for further simulator development and as a platform to conduct testing of completed satellite components and subsystems.

Once a satellite is launched, information on performance can only be obtained through a communications link. In simulated operations, the various components and subsystems employed in the satellite can be incorporated in the simulator as they are completed. This allows for constant feedback on performance throughout the development of a satellite, and gives the designers an opportunity to react to simulated operations before the system is actually deployed. Throughout this work, various components of PANSAT and the simulator are mentioned. Such references and descriptions are intended to provide the reader with a frame of reference within which to examine development of the DCS and EPS interfaces, and should not be viewed as comprehensive references for the simulator or PANSAT itself.

Chapter II discusses the objectives and components of the PANSAT project and specifically examines PANSAT's Digital Control and Electrical Power Subsystems. Chapter III introduces the PANSAT simulator and its component subsystems. Chapter IV presents the results of laboratory work performed in conjunction with this thesis. The chapter begins with a review of the *LabVIEW* program, followed by a description of the development of the simulator's Digital Control Subsystem and Electrical Power Subsystem interfaces. Chapter V offers conclusions and recommendations, including steps to be taken to accommodate anticipated changes in the simulator's configuration.

II. THE PETITE AMATEUR NAVY SATELLITE (PANSAT)

A. OVERVIEW

PANSAT is a small, spread-spectrum, store-and-forward communication satellite under development at the Naval Postgraduate School (NPS), Monterey, California. The PANSAT project was originated at the NPS by the Space System Academic Group (SSAG) to fulfill the following objectives: (1) provide practical, hands-on experience to NPS graduate students in all aspects of design, testing and operation of a small, digital, spread-spectrum, store-and-forward communication satellite; (2) demonstrate a low-cost spread spectrum system; (3) provide a compatible system for use by the amateur radio community.

Store-and-forward satellites allow asynchronous communications, as illustrated in Figure 1. PANSAT is designed to employ a spread spectrum, binary-phase-shift keying

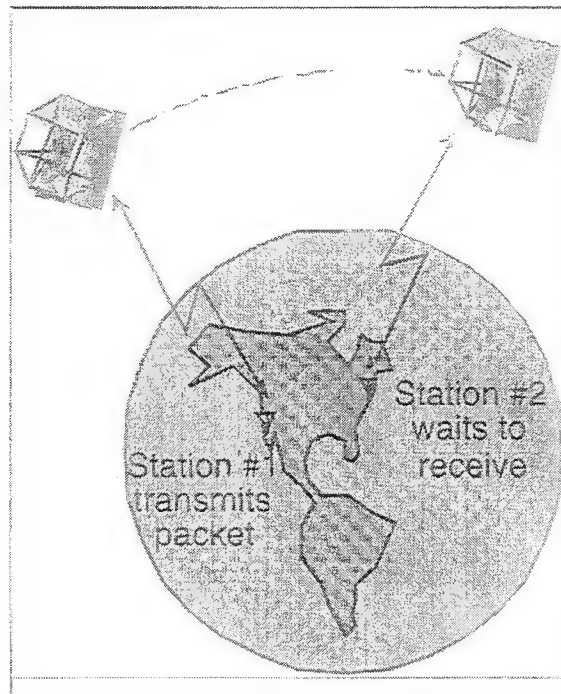


Figure 1: Store-and-forward Satellite Communication System [Ref. 1: p. 12].

(BPSK) modulated, store-and-forward, direct sequence UHF packet communication system with a center frequency of 436.5 Mhz, a bandwidth of approximately 2.5 Mhz, a data rate of 9842 bits per second, and a message storage capacity of four megabytes. The system is intended to have a mission life of two years.

PANSAT is comprised of three main spacecraft subsystems that perform electrical power, digital control, and communications functions. The satellite will be made of aluminum and will weigh 150 pounds, with an approximate diameter of 19 inches. The exterior of the satellite is composed of 18 squares, 17 of which are equipped with solar panels, and eight triangular plates. Four dipole antennas are attached in a tangential turnstile configuration to four of the triangular plates. Inside the satellite are two battery boxes, the Electrical Power Subsystem (EPS), the Communications Subsystem (COMM), and the Digital Control Subsystem. The satellite will be free-tumbling, having no attitude control or propulsion systems. [Ref. 1: p. 14]. The spacecraft's configuration is illustrated in Figure 2 and a block diagram illustrating the connections between the various satellite subsystems is presented in Figure 3. In this figure, digital signal connections, analog signal connections, and power connections are represented by thin, dotted, and thick lines, respectively.

PANSAT is scheduled to be launched into a low-Earth orbit in early 1997. Significant project milestones are plotted in Figure 4. A Get Away Special (GAS) canister will house the satellite on board the Space Shuttle. A small satellite launcher, built by Defense Systems, Inc. (DSI), will be used to push the satellite into orbit [Ref. 3: p. 5]. A typical orbit will have an altitude of 480 km at a 28.5° inclination, providing an average window time of six minutes for communication [Ref. 4: p. 1].

Once PANSAT is placed in orbit and a software uplink is established, the user will be able to send messages, receive messages that have been stored on-board the satellite, upload and download files, and read spacecraft telemetry. Through the use of

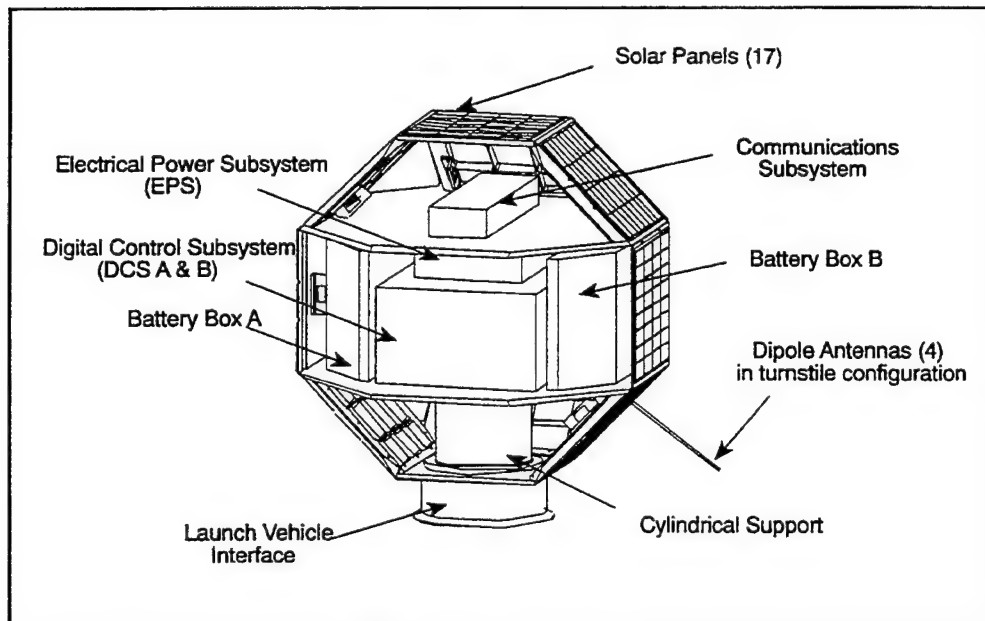


Figure 2: PANSAT Configuration [Ref. 2: p. 3].

sophisticated protocol software, multiple users will be able to establish synchronous communications links with the satellite on a single communication channel [Ref. 5: p. 4].

To communicate with PANSAT, a ground station must have a personal computer, a Terminal Node Controller (TNC), a PANSAT-specific spread spectrum modulator-demodulator (modem), and radio transmission and reception equipment. Application software for an easy-to-use interface that includes telemetry encoding will be made available by the NPS [Ref. 4: p. 2]. This software will be 100% compatible with the Surrey Satellite Technology Limited (SSTL) store-and-forward software widely used by the amateur radio community.

B. THE DIGITAL CONTROL SUBSYSTEM (DCS)

The primary functions of the DCS are to provide control of the EPS, control and operation of the COMM payload, to gather and store telemetry data, and to perform

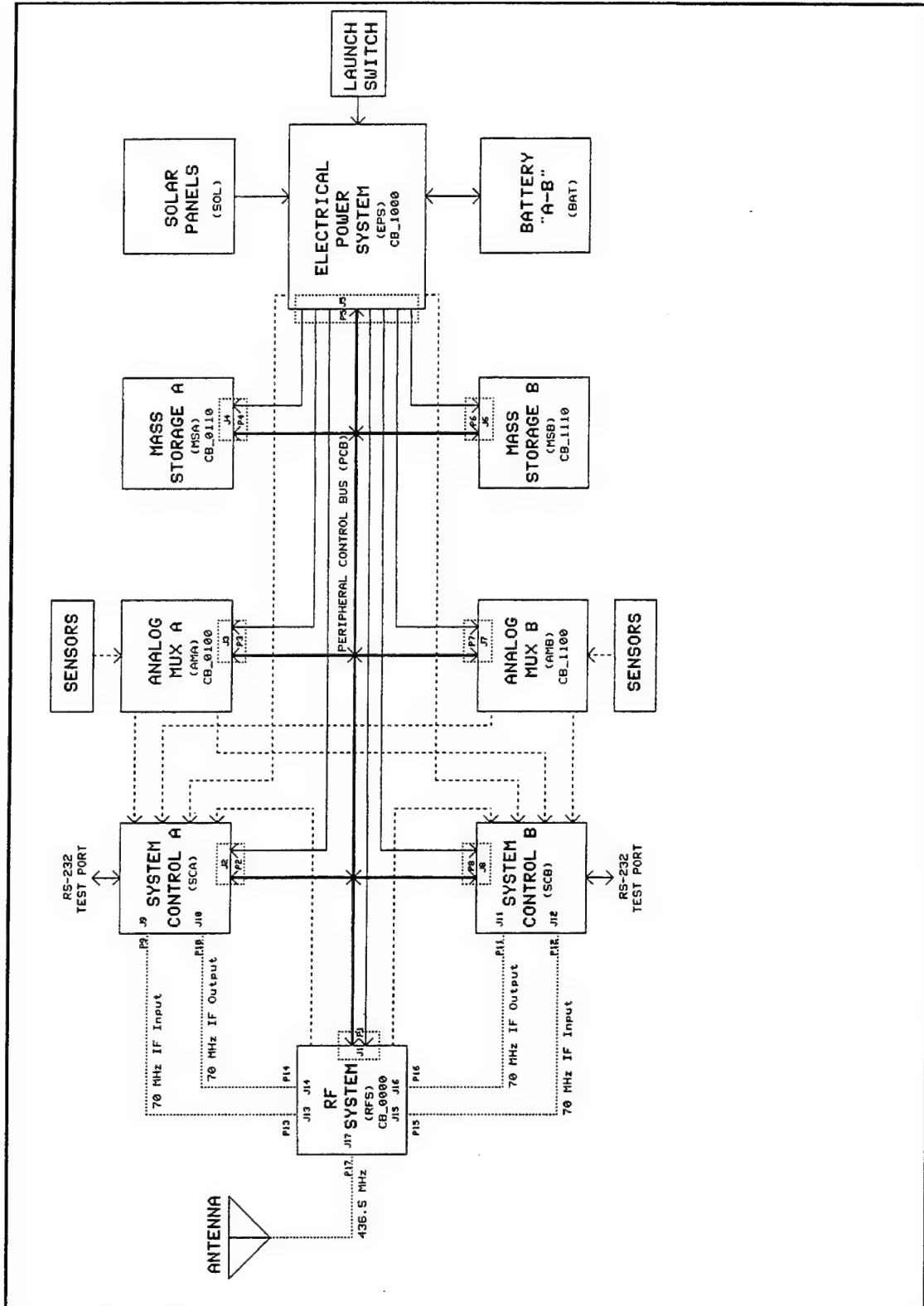


Figure 3: PANSAT Block Diagram [Ref. 6: p. 2].

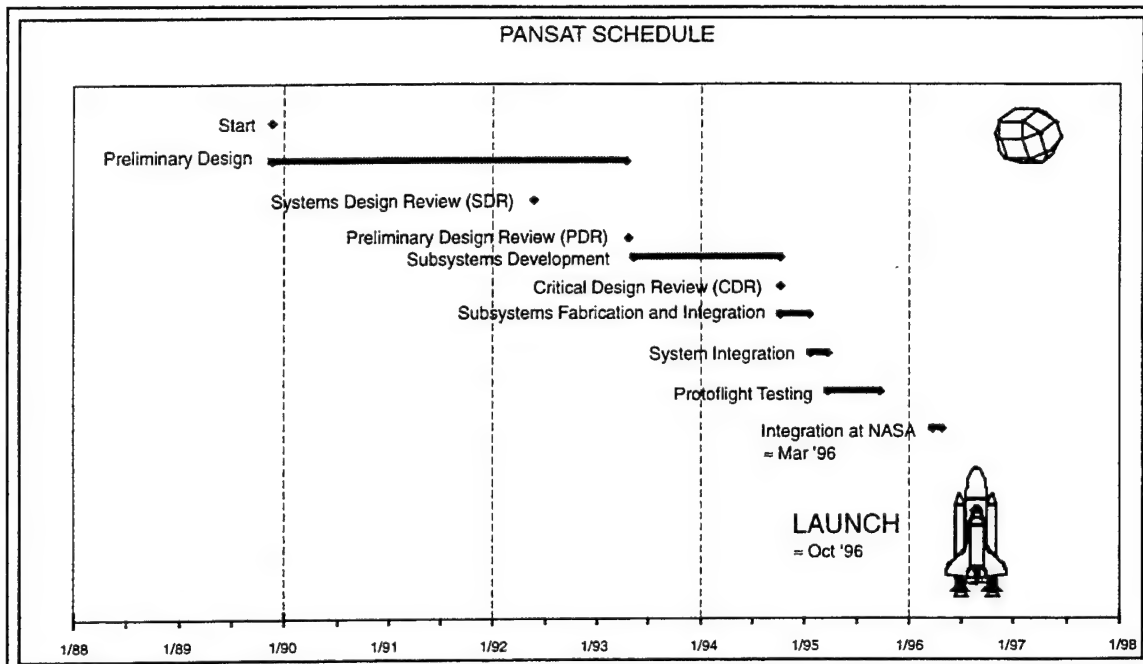


Figure 4: PANSAT Milestones [Ref. 2: p. 5].

memory management and control for message handling [Ref. 4: p. 3]. Figure 5 is a block diagram of the DCS. Digital signal connections are represented by thin lines, analog signal connections by dotted lines, and power connections by thick lines. The DCS is fully redundant, consisting of System Control Boards A and B (SCA and SCB), the analog multiplexers (AMA and AMB), and the mass storage units (MSA and MSB). If a failure should occur on one of the DCS's paired components, such as the AMA, a control board switching procedure is invoked to deactivate the failing component, in this case the AMA, and activate the redundant unit, in this case AMB.

Within each of the System Control Boards (SCA and SCB) are the following main components: The M80C186XL microprocessor, the Serial Communications Controller (SCC), the LM12454CIV analog-to-digital converter (A/D C), the 82C55 Programmable Peripheral Interface (PPI), the PA-100 Spread Spectrum Demodulator, an error-detection-and-correction (EDAC) random access memory (RAM), and 32K of programmable read-only memory (PROM). Figure 6 illustrates these components and their connections.

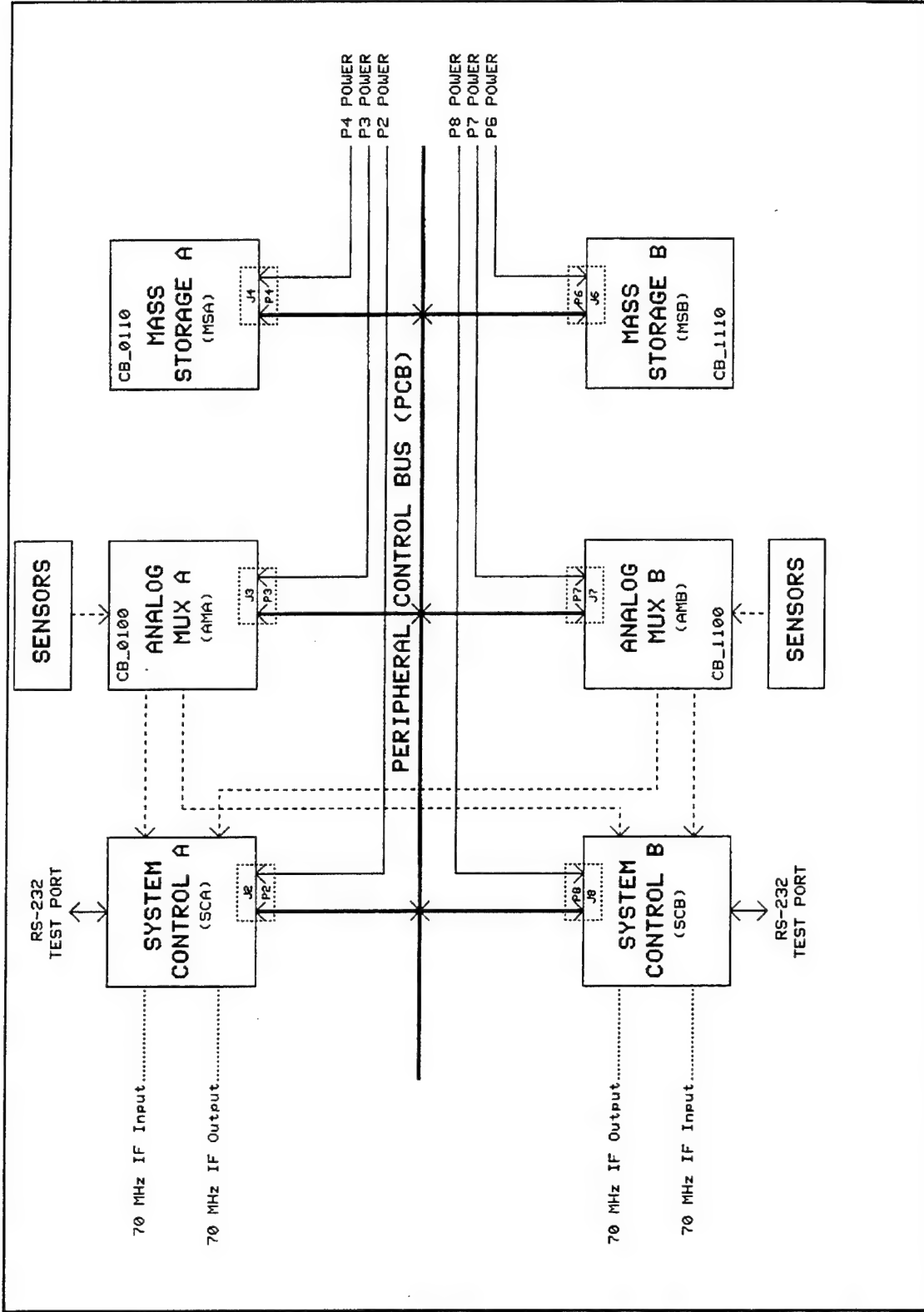


Figure 5: DCS Block Diagram [Ref. 6: p. 3].

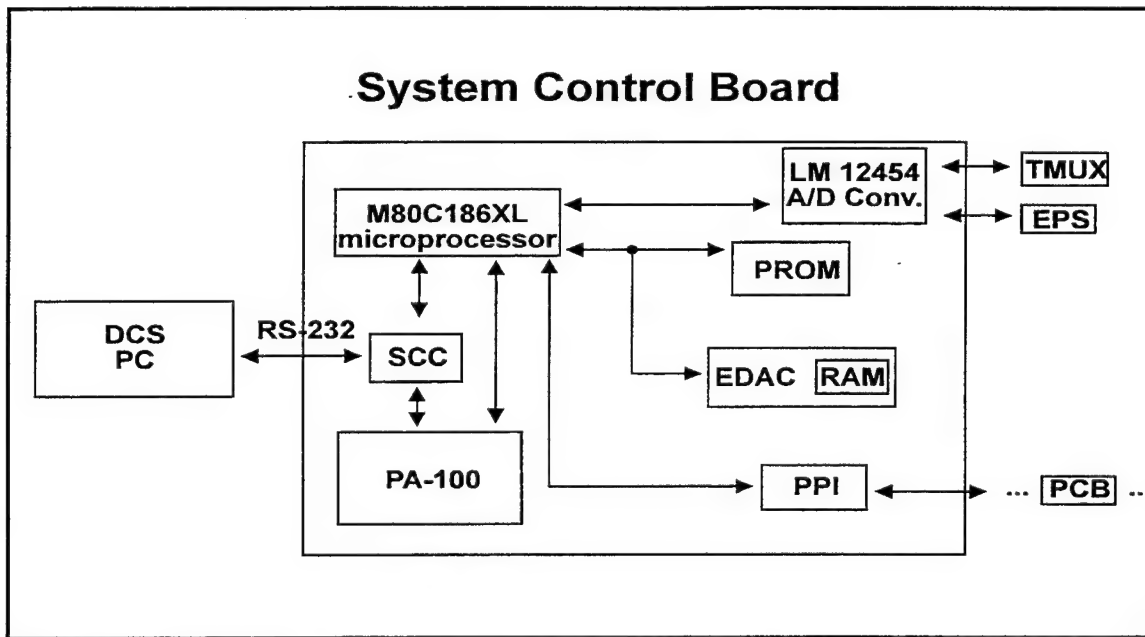


Figure 6: PANSAT System Control Board.

The M80C186XL microprocessor is responsible for handling all software activities within the spacecraft. It contains an enhanced 80186 instruction set, an interrupt control unit, a timer/counter unit, a chip select unit, a bus interface unit, and a two channel Direct Memory Access (DMA) [Ref. 7: p. 3]. The Serial Communication Controller (SCC) transfers digital data streams between the M80C186XL microprocessor and the PA-100. The microprocessor's two DMA channels are dedicated to interface with Channel A of the SCC in the synchronous mode. One DMA channel is for incoming and outgoing message traffic, while the other is used for Error Detection and Correction (EDAC) random access memory (RAM) wash. Channel B of the SCC is the PANSAT's asynchronous serial test port. The LM12454 analog-to-digital converter accepts four separate analog inputs from the multiplexer and provides one digital output to the microprocessor. It interfaces directly with the M80C186XL microprocessor chip as an I/O addressed peripheral [Ref. 7: p. 4]. The 82C55 Programmable Peripheral Interface

(PPI) is the interface between the M80C186XL microprocessor and the Peripheral Control Bus (PCB). This interface enables the System Control Boards (SCA and SCB) to communicate with and control components connected to the PCB. The PA-100 Spread Spectrum Demodulator, manufactured by PARAMAX®, interfaces between the RF analog portion of the satellite and the SCC. Its main function is the conversion of the RF analog signal to a digital signal for further processing within the microprocessor. PN code acquisition is also performed by the PA-100. Error Detection and Correction (EDAC) circuitry controls the System Control Board's random access memory (RAM). The EDAC is designed to correct single bit errors and indicate double bit errors [Ref. 7: p. 9]. The 32K PROM is directly connected to the microprocessor. It contains the programs that will bring the system online after the satellite is placed in orbit.

The analog multiplexers (AMA and AMB), also called the “temperature mux” (TMUX) due to their handling 90% of temperature signals, are directly connected to the LM12454 analog-to-digital converter via dedicated analog cables.

The two Mass Storage Units are connected to the microprocessor via the PPI and PCB. Four megabytes of memory have been allocated for each of the units, which will provide file storage for spacecraft software, user messages, and telemetry.

The Peripheral Control Bus (PCB) connects all subsystems of the spacecraft. Through the use of read and write commands, control and data flow transfer over the components connected to the PCB is orchestrated by the System Control Board. Table 1 lists the eight subsystems connected to the PCB. Currently defined PCB addresses are listed in Appendix C.

C. THE ELECTRICAL POWER SUBSYSTEM (EPS)

The primary function of the EPS is the generation and distribution of power to other subsystems in the satellite. Additionally, the EPS performs battery charge regulation (trickle, recondition, charge), multiplexes voltage and current points, and

Subsystem Name
RF Subsystem
Electrical Power Subsystem
System Control A
System Control B
Analog MUX A
Analog MUX B
Mass Storage A
Mass Storage B

Table 1: Subsystems on PCB.

gathers internal temperature measurements. Power is supplied by the EPS to other subsystems along power lines on the PCB, and is controlled by the DCS through the PCB. Current and voltage readings are gathered in the EPS and then sent to the analog-to-digital converter located in the DCS. Temperature sensors located in the EPS send measurements to the TMUX, where they are passed on to the analog-to-digital converter in the DCS. Readings from the DCS's analog-to-digital converter are sent to the microprocessor, where decisions are made regarding the regulation and conditioning of power in the EPS. The DCS sends instructions back to the EPS via the PCB, directing which EPS switches are to be enabled or disabled to regulate and condition the power. Figure 7 is a schematic diagram illustrating the flow of information to and from the EPS.

The EPS is composed of 17 solar panels, redundant nickel-cadmium Batteries A and B, a watchdog timer, temperature sensors, voltage and current muxing, power regulation and conditioning circuitry, and a launch switch that activates spacecraft startup procedures upon deployment. These components are depicted in Figure 8.

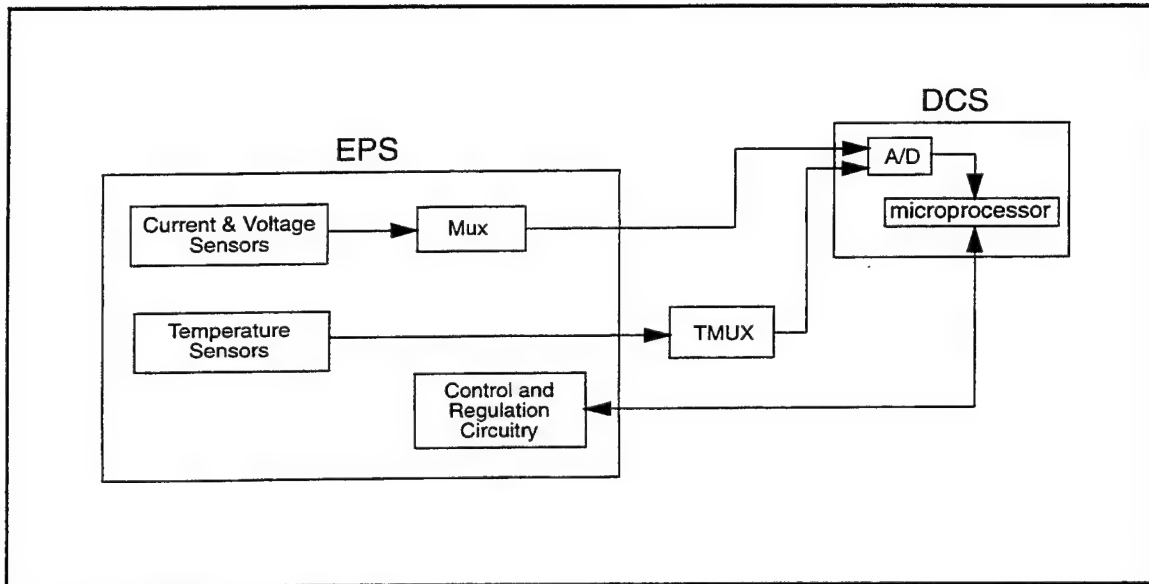


Figure 7: EPS Sensor Data Flow and Circuitry Control.

The 17 solar panels symmetrically cover the spacecraft, providing an average area of approximately 989 cm². Each panel is composed of 32 individual 1.92 x 4.00 cm silicon cells, chosen for their low cost and ability to provide ample power for spacecraft operations, connected in series. The panels were fabricated by Sectrolab, Inc., of Sylmar, California, using the K6700 silicon cell with back surface field and reflector (BSFR) [Ref. 4: p. 3].

Nickel-cadmium (NiCd) batteries were selected for use in PANSAT's EPS on the basis of their high energy density, cycle life, and excellent record of performance. Two different ratings of NiCd were considered: space-rated and terrestrial-rated. The space-rated variety is specifically designed for use in space, and is considered the most reliable, but is prohibitively priced -- each battery would cost \$100,000. Terrestrial-rated cells, on the other hand, cost \$10,000 each, and have been successfully used in space. The PANSAT project will utilize a total of six of the latter; four flight and two prototype 12 volt batteries will be produced. It is anticipated that the batteries may be depleted beyond viable levels of operation upon deployment from the Space Shuttle. While the spacecraft

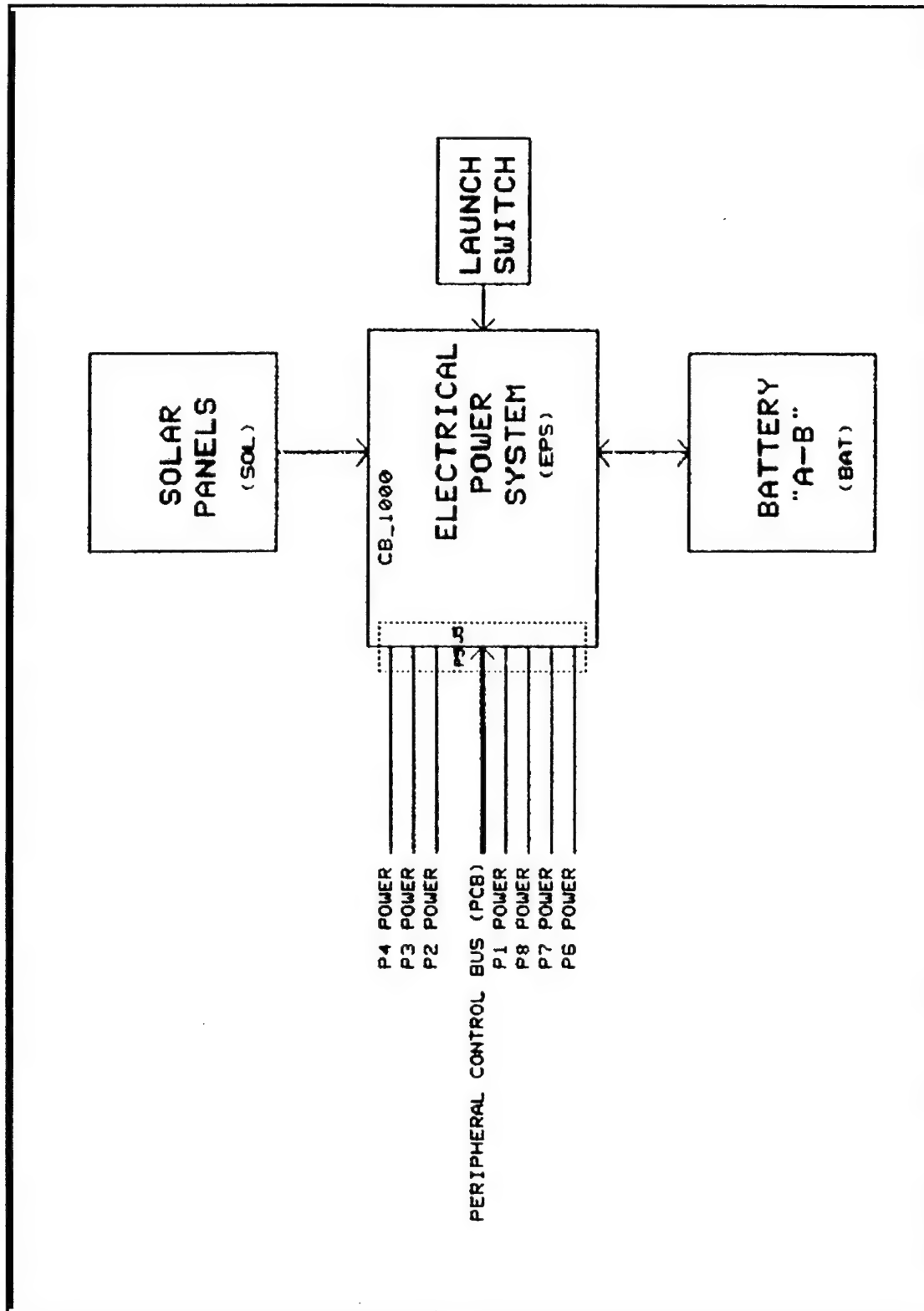


Figure 8: PANSAT Electrical Power Subsystem [Ref. 6: p. 4].

is operating in sunlight, however, trickle charging of the batteries will occur. Once workable voltage levels are reached in the battery cells, the other subsystems will begin to power up.

Three types of sensors are necessary for the EPS to regulate and condition power throughout the satellite. Current sensors will monitor solar panel and battery charge and discharge. Using specially designed precision resistors for current sensing, these sensors will determine PANSAT's orientation and roll rates by monitoring the currents of eight of the solar panels. Voltage sensors will monitor Batteries A and B, the PCB, and all of the solar cells, in accordance with a suggestion by the Naval Research Lab. Twenty voltage measurement points will be muxed in the EPS, allowing monitoring of all of the solar cells. Two temperature sensors will be located on each of the two battery cells. Temperature sensing diodes have been chosen for their linearity and accuracy. The addition of a differentially ended temperature sensor to each battery cell is still being contemplated by the project staff [Ref. 8: p. 8].

The watchdog timer consists of a CD4060BMJ-MIL 14 bit ripple carry binary counter, "flip-flops" and logic gates, a capacitor, and resistors. The watchdog timer allows control of the satellite to be regained if the active System Control Board (SCB) locks up. Each minute that the SCB is functioning properly, it sends a signal to the watchdog timer that resets the timer to zero; if the SCB has malfunctioned, no signal is sent. When the watchdog timer reaches six minutes, power to the active SCB is discontinued, and power is routed to the alternate SCB.

III. THE PANSAT SIMULATOR

A. OVERVIEW

The simulator's physical configuration will eventually be identical to that of the satellite. At this time, however, neither the Electrical Power Subsystem (EPS) nor the Communication Subsystem (COMM) is connected to, or functioning on, the embedded system. The operation of those two subsystems was not essential to the development of the DCS interface, illustrating the user's ability to evaluate completed components without the benefit of a complete and fully operational system. Figure 9 shows the eventual configuration of the simulator's subsystems, while Figure 10 illustrates how additional PC inputs are incorporated to simulate the EPS and COMM Subsystems.

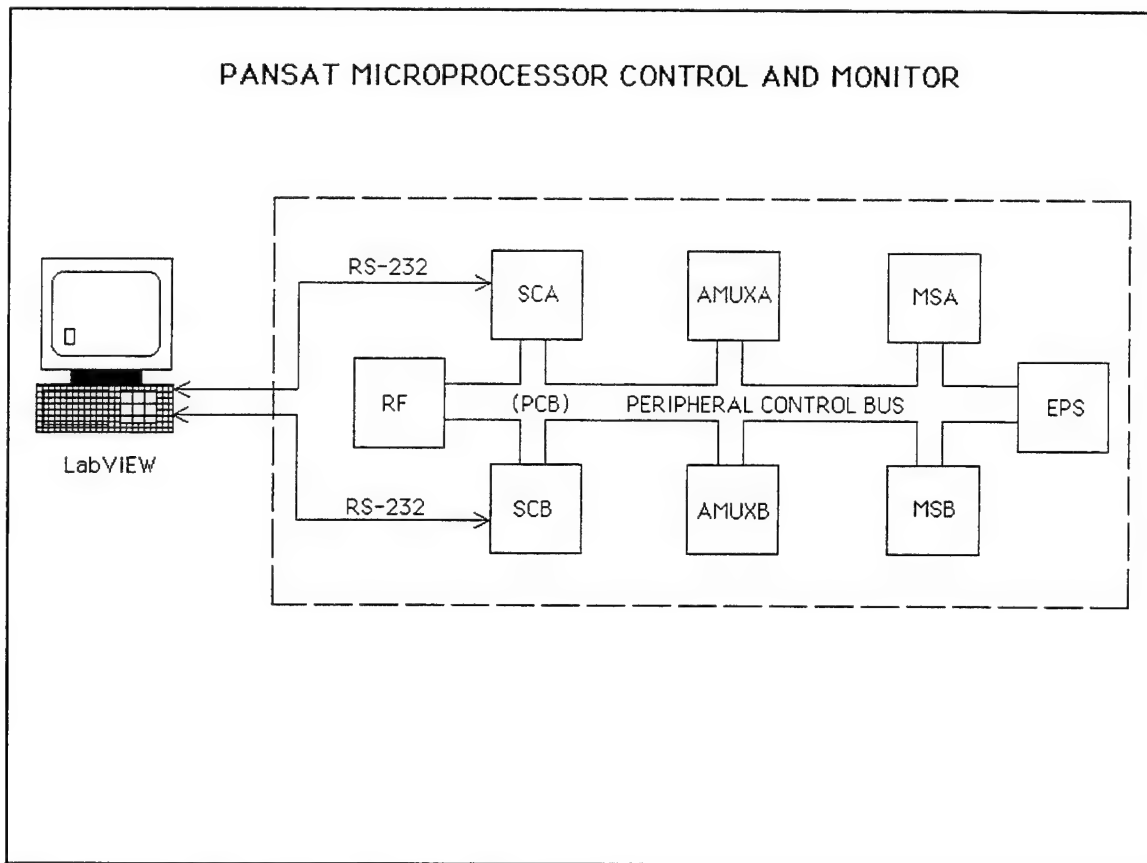


Figure 9: PANSAT Simulation Configuration.

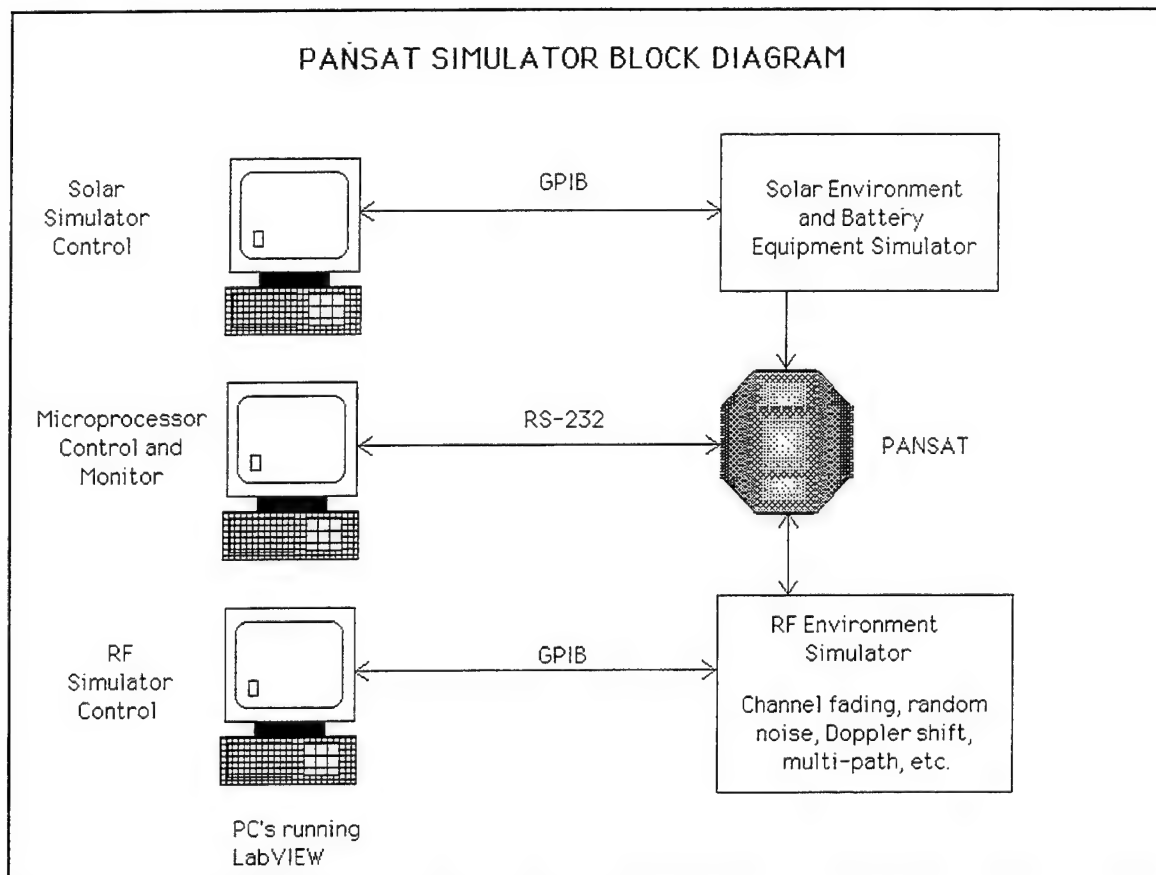


Figure 10: PANSAT Simulator Block Diagram.

Use of the simulator will allow:

1. subsystem developers to test the functionality of designed components while the other system components are yet incomplete;
2. prelaunch testing of protoboard design;
3. pretesting of possible configuration changes to the spacecraft after it has been launched;

4. analysis of particular circumstances that the satellite may encounter while in orbit;
5. duplication of previously encountered circumstances for real time analysis, and;
6. student exposure to a powerful educational tool.

B. THE DIGITAL CONTROL SUBSYSTEM (DCS)

Functionally, the simulator's DCS is the same as that of the spacecraft. Simulation of DCS operation was accomplished using the Space Thermal Acoustic Refrigerator's (STAR) System Control Board. Additionally, the embedded system's DCS is composed of the Peripheral Control Bus (PCB), and one analog multiplexer. The PCB is identical to that used in PANSAT. At present, however, there is no redundant capability in the simulator's DCS, the analog multiplexer is still undergoing testing, and there is no mass storage unit connected to the simulator. These differences had no effect on the development of the DCS interface. Figure 11 illustrates the configuration of the connection for microprocessor control of the PANSAT simulator.

The STAR System Control Board is less sophisticated than PANSAT's System Control Board, but fulfills the same functional purpose. The STAR System Control Board consists of several components. The 8088 Microprocessor is used in place of PANSAT's M80C186XL, the Universal Asynchronous Receiver/Transmitter (UART) 8252 is used in place of PANSAT's Serial Communication Controller, and the HP3478A multimeter fulfills the function of PANSAT's LM12454 analog-to-digital converter. Unlike the LM12454, which has four separate analog inputs from the multiplexer, the HP3478A multimeter accepts only a single analog input. This difference can be compensated for by stacking four multimeters on top of one another and feeding the four separate analog inputs into a corresponding multimeter (resulting in an analog-to-digital

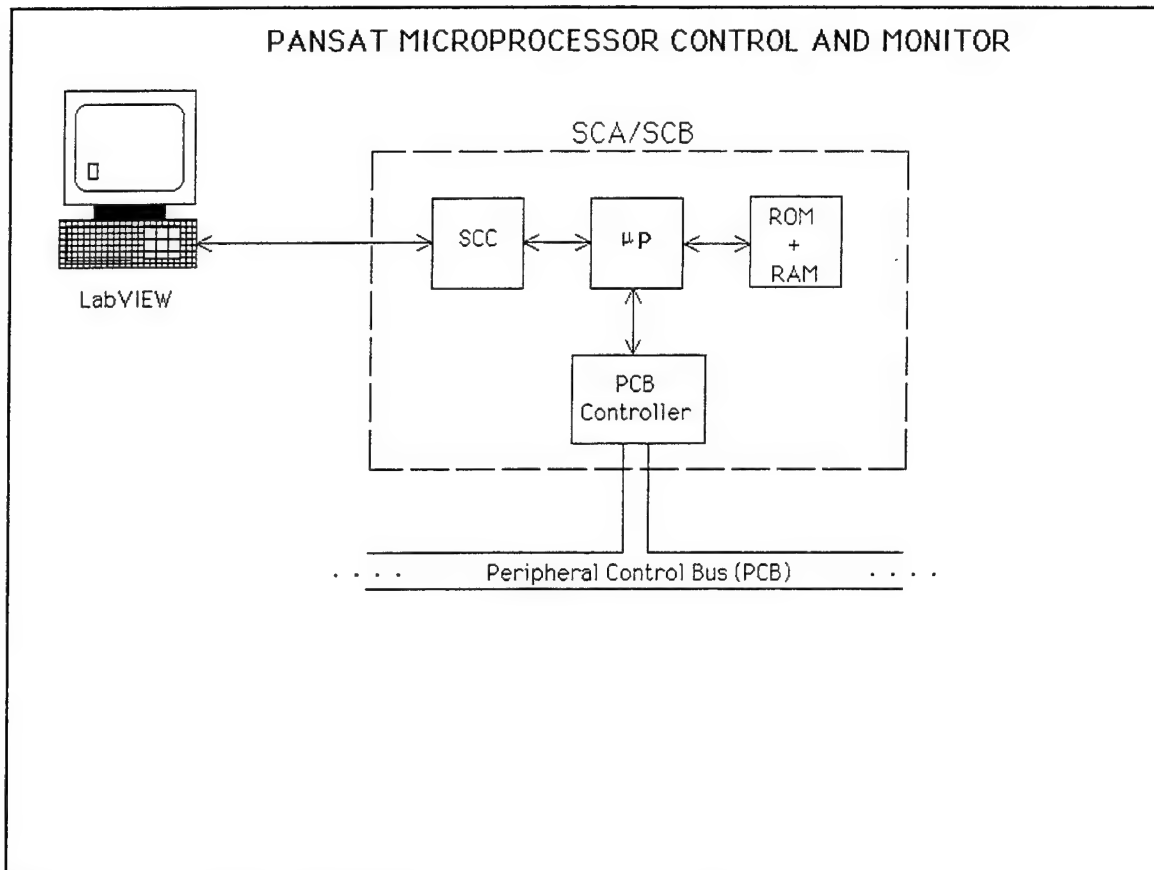


Figure 11: PANSAT Microprocessor Control Connection.

conversion outside of the control board's physical configuration).

Digital output to the simulator PC is provided via the General Purpose Interface Bus (GPIB), which also provides the interface for PC control of the HP3478A multimeters. As in PANSAT, an 82C55 Programmable Peripheral Interface (PPI) is used. To allow it to function properly, however, the STAR System Control Board was modified (most notably, the hardware handshake lines were connected). Additionally, the simulator does not currently employ a PA-100 Spread Spectrum Demodulator, although design detail and integration into the embedded system is currently underway.

There is 16K of read only memory (ROM) and 64K of random access memory (RAM) connected directly to the microprocessor on the STAR System Control Board.

The ROM is used to store the software necessary to initialize the electronics and maintain the serial link that allows commands and data to be passed from the user to the embedded system. The RAM is used to store the data that allows interaction between the user and the embedded system, and also allows new software to be uploaded via the embedded system's RS-232 port and run in place of the software in ROM.

C. THE ELECTRICAL POWER SUBSYSTEM (EPS)

The simulator EPS is functionally the same as that of the spacecraft. It is intended that power use duty cycles will be examined and testing on best and worst case eclipse percentages will be performed to test EPS response. The eventual physical configuration of the simulator EPS will be identical to that of the spacecraft's, except that the simulator will not incorporate solar panels, which will be simulated by an HP6653A Power Supply. Figure 12 illustrates how the system will simulate the generation of power in a regular orbit cycle. Prototype NiCd batteries will be integrated into the embedded system once they arrive from the manufacturer, making it possible to test their performance. Until then, 12 volts of power is being provided to the DCS and PCB by an HP6216A DC Power Supply. The simulator will also employ the same current, voltage, and temperature sensors as found in the spacecraft. Temperature sensors are currently undergoing testing. Until then, measured voltage and current are being read by the HP6653A Power Supply and the various multimeters employed in the simulator. A watchdog timer identical to that on the spacecraft is currently operational on the embedded simulator system, connected to the PCB.

D. THE COMMUNICATIONS SUBSYSTEM (COMM)

Once the PANSAT Control Board replaces the STAR Control Board, the implementation of the Communications Subsystem within the rest of the simulator can

begin. Until then, the operation of COMM is achieved using a PC with a GPIB and readily available equipment, as shown in Figure 13.

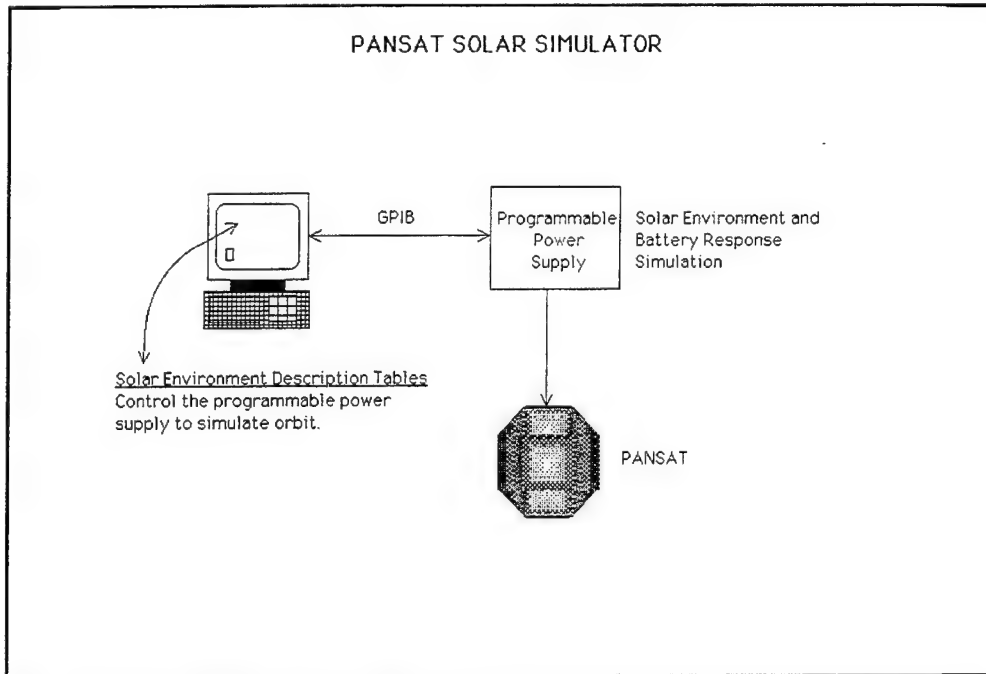


Figure 12: PANSAT Solar Simulator.

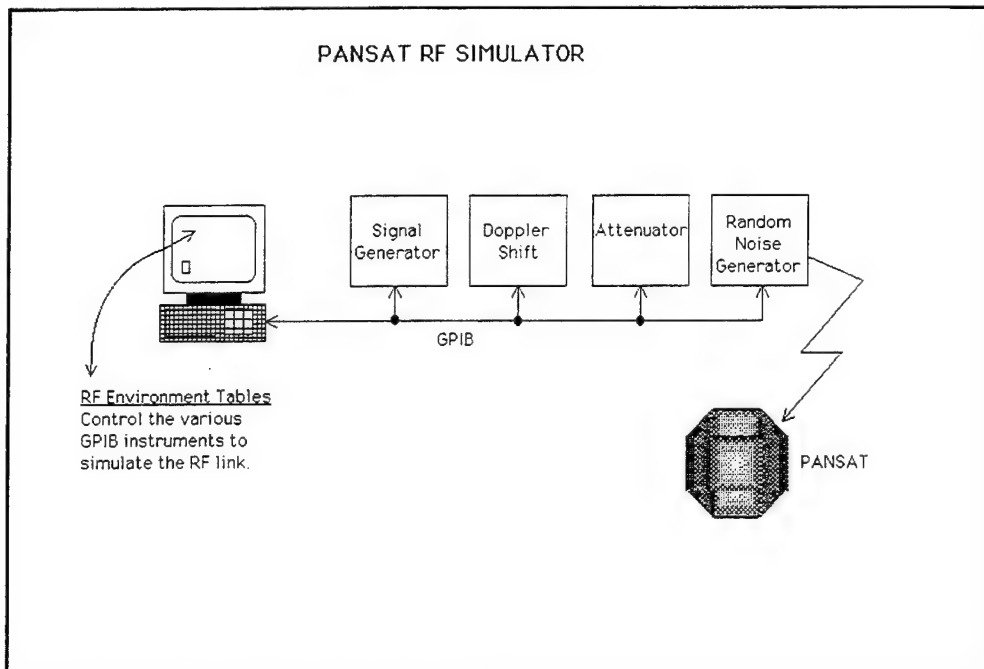


Figure 13: PANSAT RF Simulator.

IV. INTERFACE DEVELOPMENT

A. LABORATORY VIRTUAL INSTRUMENT ENGINEERING WORKBENCH (LabVIEW)

Graphical programming uses graphic symbols rather than text to create and describe algorithms. These symbols are connected within block diagrams to carry out specific functions, allowing the user with little programming experience to develop complex programs. Symbols, terminology, icons, and ideas are expressed in a format familiar to scientists and engineers.

1. LabVIEW

LabVIEW is a software package that uses graphical programming language to create what are known as *virtual instruments*. A virtual instrument (VI) is actually a LabVIEW program that has the appearance and operational characteristics of a real instrument when displayed on a computer monitor. Each VI program includes a *front panel*, a *block diagram*, and an *icon and connector*. Additionally, LabVIEW has extensive libraries containing functions and subroutines for most programming tasks, including data acquisition and instrument control, and is equipped with trouble-shooting tools such as break points, single-step through, and impulse step through.

The front panel is the portion of the program with which the user interacts, and is pictured in Figure 14. The front panel simulates the panel of a physical instrument, and can be designed to suit the user's preferences. Knobs, push buttons, graphic displays, terminal screens, and other controls and indicators can be simulated in a VI, while data or commands are input via computer keyboard or mouse. The block diagram utilizes the graphic Programming Language G to express instructions, as shown in Figure 15. Note that VI *Volt Read* is considered a *subVI* located within the VI *Digital Thermometer*. Block diagrams are similar in appearance to the circuit diagram of an instrument, with

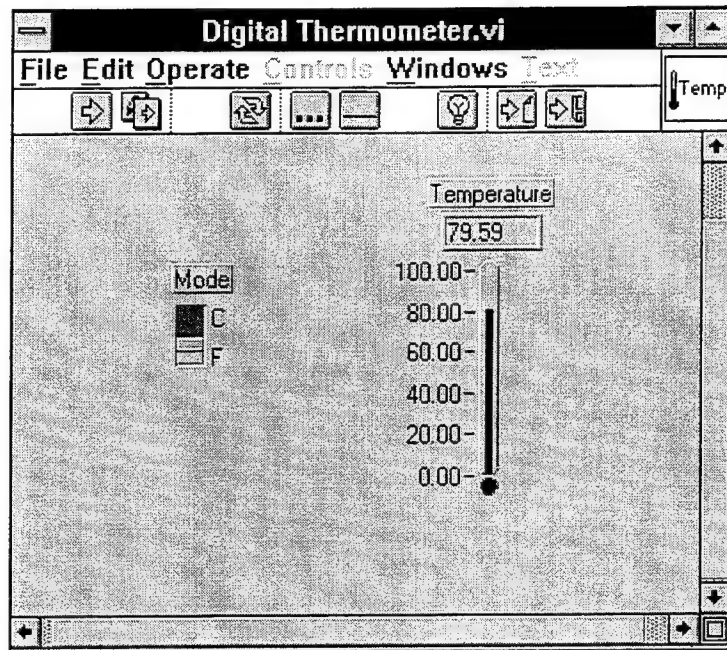


Figure 14: *Front Panel for VI Digital Thermometer.*

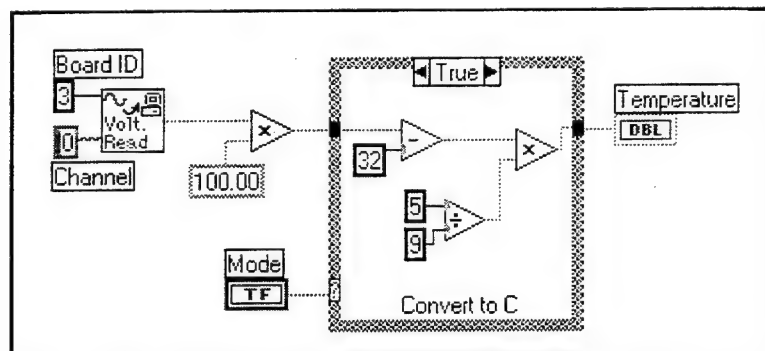


Figure 15: *Block Diagram for VI Digital Thermometer.*

data flowing between each function through the lines that connect them. Block diagrams represent a pictorial solution to a programming problem, and are the source code for the VI. The icon and connector of a VI depicts a graphical parameter list, as shown in Figure 16. The VI is graphically represented by the icon, while the connectors represent its input and output terminals; data flows between VIs via the connectors shown on an icon.

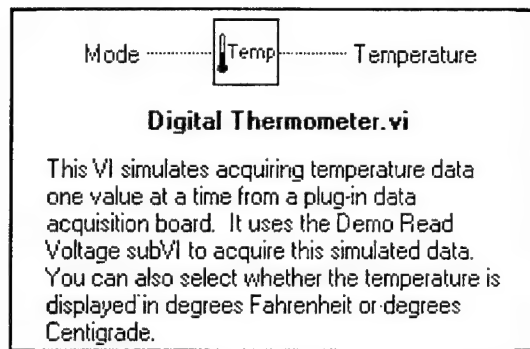


Figure 16: *Icon and Connectors for VI Digital Thermometer.*

2. Serial Port Communication

Communication between the PC and the Digital Control System (DCS) is accomplished through the use of the simulator's serial port. In developing the DCS interface, it was essential to identify suitable VIs for serial port control. Several of these VIs are discussed below [Ref. 9: pp. 4-1-4-6].

The VI *Open Serial Driver* is used to initialize a selected serial port, and is included as a subVI in each of the following descriptions. The icon and connectors for *Open Serial Driver* are pictured in Figure 17. The connector shown to the left of the icon is a control, where input is determined by the user. The connectors on the right are indicators, and show results following execution of the VI. If an input error had occurred, for example, the error would be shown on the input error connector. It is standard in LabVIEW for control connectors to appear on the left and indicators on the right. The VI *Bytes at Serial Port*, shown in Figure 18, determines the number of bytes in the input buffer of the serial port, and is a subVI of *Serial Read* and *Serial Read with Timeout*. Figure 19 shows the icon and connectors for *Serial Port Write*. Note that strings written to the computer are input on the connector labeled "string to write" and sent out over the designated serial port. Shown in

Figure 20, the VI *Serial Port Read* returns the number of characters that the “requested byte count” connector specifies. *Serial Read with Timeout* reads the requested byte count (unless a timeout condition is met first), as shown in Figure 21. A timeout error is indicated by the “Timeout” connector if the requested bytes do not arrive at the serial port buffer before a specified time.

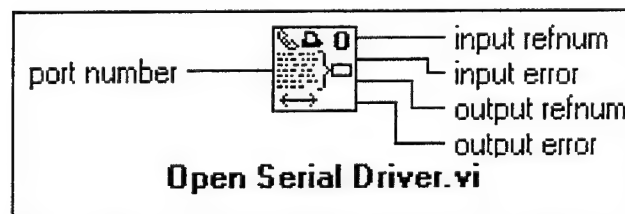


Figure 17: *Icon and Connectors for VI Open Serial Driver.*

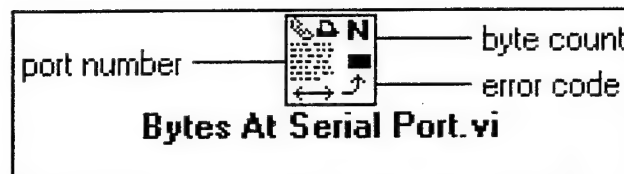


Figure 18: *Icon and Connectors for VI Bytes at Serial Port.*

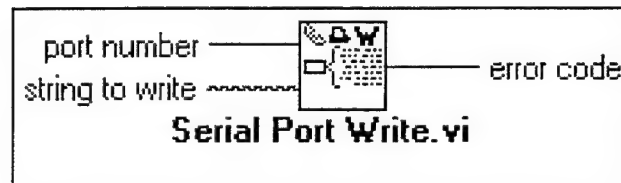


Figure 19: *Icon and Connectors for VI Serial Port Write.*

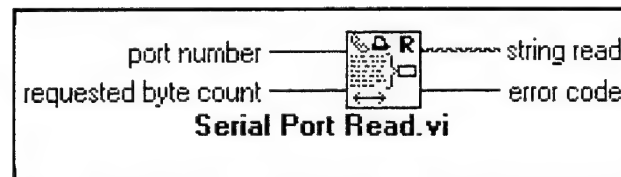


Figure 20: *Icon and Connectors for VI Serial Port Read.*

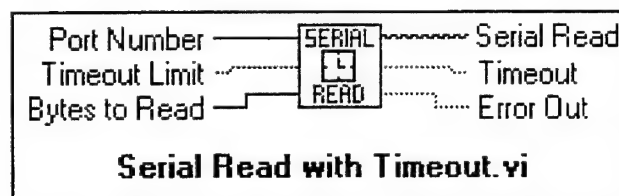


Figure 21: *Icon and Connectors for VI Serial Read with Timeout.*

3. General Purpose Interface Bus (GPIB)

The GPIB is an interface system through which the interconnected electronic devices of the simulator communicate. The GPIB is used to provide simultaneous computer control of test and measurement instruments such as the HP3478A multimeter, the HP6653A power supply, and the HP 6060A electronic load. At this time, the multimeter is being used as an analog-to-digital converter. The power supply simulates the solar panels and the electronic load simulates loads applied to the EPS by satellite operation.

An IEEE 488.2 industry standard AT-GPIB board was installed in the simulator PC, and NI-488.2 software was installed on the PC's hard disk drive under the AT-GPIB directory. During the installation process, the line "device-\at-gpib\gpib.com", which activates the AT-GPIB driver each time the PC is started, is inserted in the systems "config.sys" file. For more information on the GPIB and NI-488.2 software, see References 10 and 11.

There are two GPIB VIs that are prominent in the operation of GPIB controlled devices. *GPIB Write* writes commands to the addressed device. As shown in Figure 22, "address string" is the address of the device being written to, "data" is the command string being sent to the addressed device, "mode (0)" indicates how to terminate the *GPIB Write*, with a default mode of "0", "timeout ms ..." is the time allowed to complete the operation, with a default value of "488.2 global" (if operations are not completed in the specified time, operations cease), and "status" indicates the GPIB controller status after the write operation. The icon and connector for *GPIB Read* is pictured in Figure 23. In this VI, "address string" identifies the address of the device to be read, "byte count" identifies how many bytes the user wants read, and "data string" is the data that is read from the specified device. The contents of the data string are determined by the write command that the user has given the addressed device prior to executing the read

command. Termination of this VI occurs when any one of the following events occur [Ref. 12: pp. 8-10]:

1. The number of bytes requested by the user is met
2. An error is detected
3. The specified time limit is exceeded
4. The end of message character is detected
5. The end of string character is detected

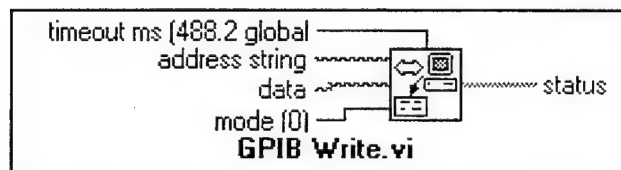


Figure 22: *Icon and Connectors for VI GPIB Write.*

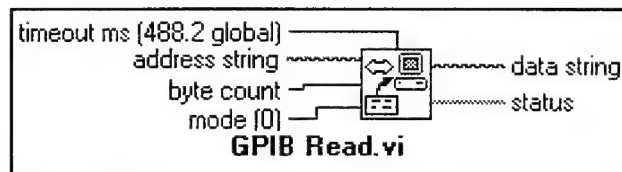


Figure 23: *Icon and Connectors for VI GPIB Read.*

B. DCS INTERFACE

1. Ground Level Protocol

Communications are transmitted through the serial port in *packets*. A packet consists of a flag byte, two length bytes, data of user-determined length, and two check sum bytes for cyclic redundancy check (CRC), as illustrated in Table 2. "Flag", 7E hexadecimal, identifies the beginning of the packet, "Len1" and "Len2" together represent the length in bytes of the data field, "Data" is the actual message or command to be executed, and "CRC1" and "CRC2" are used together to determine if an error in transmission has occurred. The peculiar order of "Len1" and "Len2" is due to the byte ordering of the Intel 80186 architecture where the low byte always proceeds the high byte. The VI used to format a message as shown in Table 2 is called *Send*, while a separate VI, *RCVR*, is used to reverse the process and perform an error check on data coming back to the PC from the embedded system. For the code and a brief description of *Send* and *RCVR*, see Appendix A.

Flag (1 byte)	Len1 (least significant byte)	Len2 (most significant byte)	Data (user determined)	CRC1 (1 byte)	CRC2 (1 byte)
------------------	----------------------------------	---------------------------------	---------------------------	------------------	------------------

Table 2: Serial Port *Packet*.

Within the "data" field is the "command data" field, which occupies the first two bytes in the data field and is logically handled in the order shown in Table 3, where each column is a nibble, or four bits, in length. Table 4 illustrates the order in which the command data field is actually received by the microprocessor. The unusual order, known as Little Endian, is due to the 80186 architecture. Table 5 shows the meaning of each bit within the data field flag. For a command being sent out from the microprocessor, the most significant bit of the "data field flag" is set. For a command received into the microprocessor, this bit is zero. The second bit in the data field flag is

an error bit, signifying an error if it is set. The third and fourth bits of the flag data field are undefined, reserved for possible future use. “Command 2”, “Command 1”, and “Command 0” in Tables 3 and 4 list the hexadecimal equivalent of the low level commands processed by the system. Following the command data field is the data representing the low level commands that are supported by the simulator. There are a total of eight low level commands that are supported by the embedded system; their hexadecimal equivalents range from 000 through 007.

Data Field Flag	Command '2'	Command '1'	Command '0'
-----------------	-------------	-------------	-------------

Table 3: Logical Order of Command Data Field.

nibble '0'	nibble '1'	nibble '2'	nibble '3'
Command '2' (most significant)	Data Field Flag	Command '0' (least significant)	Command '1'

Table 4: Command Data Field.

Bit Position	Function
_xxx (most significant bit)	Command Send or Acknowledgement
x_xx	Error
xx_x	undefined
xxx_ (least significant bit)	undefined

Table 5: Bits of the Command Data Field Flag Defined.

2. Low Level Commands

Table 6 shows the low level commands currently employed in the simulator, along with the syntax to be employed for each. The first column lists the command, the second column lists the user syntax format, and the third column lists the corresponding hexadecimal representation for the command. These commands are text-based, and are entered into the embedded system by the user via the PC interface. The ranges of CPU addresses and data fields employed are presented in Table 7.

Command	User Syntax	Hexadecimal Representation (12 Bits)
Memory Read (8-bit)	<i>R8 cpu_address</i>	000
Memory Write (8-bit)	<i>W8 cpu_address data8</i>	001
Memory Read (16-bit)	<i>R16 cpu_address</i>	002
Memory Write (16-bit)	<i>W16 cpu_address data16</i>	003
I/O Port Read	<i>IN io_port</i>	004
I/O Port Write	<i>OUT io_port data8</i>	005
PCB Read	<i>PCBR dev_address sub_address</i>	006
PCB Write	<i>PCBW dev_address sub_address data8</i>	007

Table 6: Low Level Commands.

Parameter	Range (hex)	Description
<i>cpu_address</i>	0 - FFFFF	CPU address for the ROM and RAM
<i>io_port</i>	0 - FFFF	CPU I/O Port
<i>pcb_address</i>	0 - F	Peripheral Control Bus Device Address
<i>sub_address</i>	0 - 3	Peripheral Control Bus Device-Subaddress
<i>data8</i>	0 - FF	8-bit data value
<i>data16</i>	0 - FFFF	16-bit data value

Table 7: Address and Data Conventions.

“Memory Read (8-bit)” is used to read the contents of the specified CPU address and to address the embedded system’s ROM or RAM, which will contain message, code, data for code, and telemetry information. The command’s syntax to the DCS is shown in Table 6; the information received back from the DCS is in the format *R8 cpu_address data8*, where “data8” is the requested information, which can be separated and further processed in the LabVIEW environment, and is one byte in length. “Memory Write (8-bit)” is used to write eight bits of data to the specified CPU address. To verify that the command was executed, the computer returns the command to the user in the same syntax as it was received.

“Memory Read (16-bit)” and “Memory Write (16-bit)” both follow the same conventions as the 8-bit commands above, except that the 16-bit commands read and write 16, rather than eight, bits of data. The LabVIEW interface makes the swap between low and high data bytes that is necessitated by the architecture of the 80186, allowing the user to send and receive the data in normal order.

“I/O Port Read” is used to read components connected to the I/O ports of the M80C186XL microprocessor. These components include the SCC, the PA-100, the LM12454 A/D converter, the EDAC controller, and the PPI, which drives the PCB. The information received from the DCS as a result of this command’s execution is in the

format *IN io_port data8*, where “data8” is one byte in length. “I/O Port Write” is used to write eight bits of data to the components with I/O ports on the microprocessor. The microprocessor repeats the command back to the user to acknowledge that it has been executed.

“PCB Read” is used to determine the status of components connected to the PCB. Information is sent to the user from the DCS in the format *PCBR dev_address sub_address data8*. “PCB Write” is used to send control commands to the components connected to the PCB. As with the other write commands above, the DCS repeats the command back to the user to acknowledge execution.

3. Control

Each of the commands discussed above is executed from within a LabVIEW VI called *Control*. The icon and connectors for *Control* are pictured in Figure 24 and the front panel is presented in Figure 25, while Appendix A contains a block diagram of the VI. In Figure 25, “Command String In” allows the user to type any of these low level commands, at which time the system carries out the command and returns a response to the “Command Reponse”. Errors are indicated by the “Serial Write”, “Command”, and “Response Error” buttons, which darken if an error occurs.

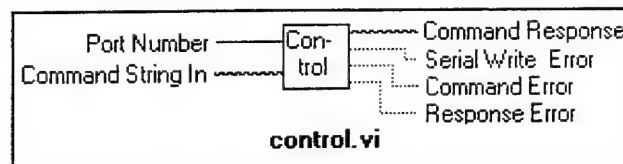


Figure 24: *Icon and Connectors for VI Control.*

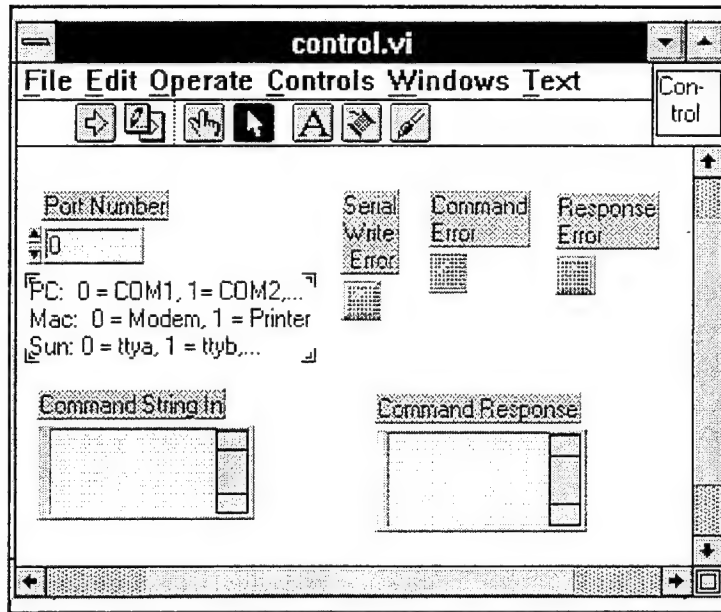


Figure 25: *Front Panel for VI Control.*

4. High Level Commands

The high level commands that are currently supported by the system are listed in Table 8. As further needs are identified, more high level commands can be added. “Run” will open a user created file that performs a series of low level commands. Such a user file can be created using a text editor such as Microsoft® Notepad™ and stored in the path “C:\pansat\simulato\run”. Useful in performing a series of commands that are repetitive, lengthy, or often used, this command actually executes within the LabVIEW environment, sending each individual command to the simulator’s microprocessor as a single command line and storing each response for display to the user. User files are handled from a VI called *File Handler*; the icon and connectors for *File Handler* are pictured in Figure 26. For further information on *File Handler*, consult Appendix A.

Brief Command Description	User Syntax	Hexadecimal Representation (12 Bits)
Run a File	RUN <i>filename</i>	008
Wait (ms)	WAIT <i>time-in-ms</i>	009
Enable Embedded System	ENABLE	00A
Disable Embedded System	DISABLE	00B
Comment	;	00C
A/D Converter Read	ADR <i>ad_channel</i>	00D

Table 8: High Level Commands.

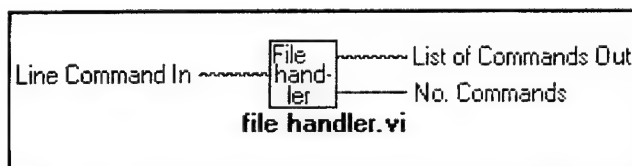


Figure 26: Icon and Connectors for VI File Handler.

“Wait” is a command used to pause system operation. As with “Run” above, “Wait” commands are not issued to the simulator’s microprocessor, but are handled directly by LabVIEW using the simulator PC’s clock. The time parameter for this command is entered in milliseconds, and the command is not repeated back to the user upon execution. The “Wait” command is administered through a VI called *Multi-Wait*, pictured as an icon and connectors in Figure 27, and as a block diagram in Appendix A.

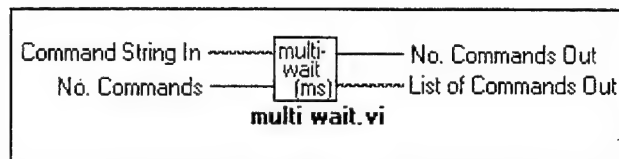


Figure 27: *Icon and Connectors for VI Multi Wait.*

The “Enable” and “Disable” commands are used to activate and deactivate the simulator’s microprocessor, respectively. These commands are repeated back to the user to acknowledge execution. Useful in the development of the simulator interface, these commands allow the LabVIEW software to be run without having the microprocessor to send commands to the various simulator components on the PCB (disable mode). As physical components are designed and implemented for testing, the microprocessor can be activated (enable mode), allowing the components’ performance to be evaluated. Another useful command is “Comment”, which allows comments to be attached to a file or window. In accordance with common programming conventions, the comment is inserted in the file or window preceeded by a semicolon. This command is administered by the *VI Comment Handler*, pictured as an icon and connectors in Figure 28 and described in more detail in Appendix A.

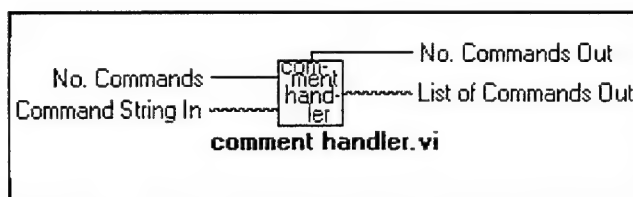


Figure 28: *Icon and Connectors for VI Comment Handler.*

Eventually, the PANSAT System Control Board will replace the STAR System Control Board, allowing the command “A/D Converter Read” to function through the simulator’s microprocessor. An intermediary procedure has been developed to execute this command through the GPIB and using a multimeter. The variable *ad_channel* is a hexadecimal number ranging from “0” to “3” that represents the channel of the A/D Converter to be read. When the PANSAT SCB is in place, data will be returned from the microprocessor in the format *ad_channel data16*. Meanwhile, readings from the multimeter are returned to the user through the GPIB and the use of a VI called *ADR Fake*, which is pictured as an icon and connectors in Figure 28. For more information on *ADR Fake*, see Appendix A.

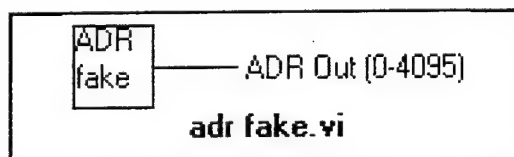


Figure 29: *Icon and Connectors for VI ADR Fake.*

The high level commands presented here are all run through a VI called *Multi Line Control*, which is shown as an icon and connectors in Figure 30, and discussed in further detail in Appendix A.

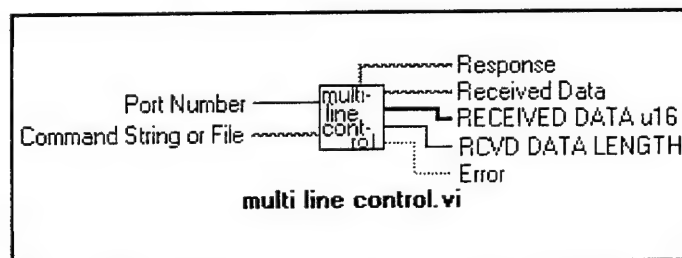


Figure 30: *Icon and Connectors for VI Multiline Control.*

5. TSWEET2

TSWEET2 is a VI application used to test the operation of the simulator's TMUX. Its front panel is shown in Figure 31, where the temperature array is plotted on the "Graph Out" portion of the panel. To start the application, the user "clicks" on the arrow button located in the upper left portion of the panel and waits for results to be displayed.

Figure 32 displays the block diagram for *TSWEET2*. When the application is started, the file "TMUX1" (see Figure 33) is run. This file contains a series of commands specifying that sensors S-1 through S-16 be read, and employs the various commands, such as "Enable", "Comment", "Wait", "PCBW", and "ADR", that have been discussed thus far. The VI *Multiline Control* processes the information, sending each "PCBW" command to the simulator's TMUX. A "Wait" of 50 milliseconds then occurs, followed by an "ADR" command. Using the GPIB to control the multimeter attached to the simulator, voltage readings take place. Measurements are then sent to the VI *Convert to*

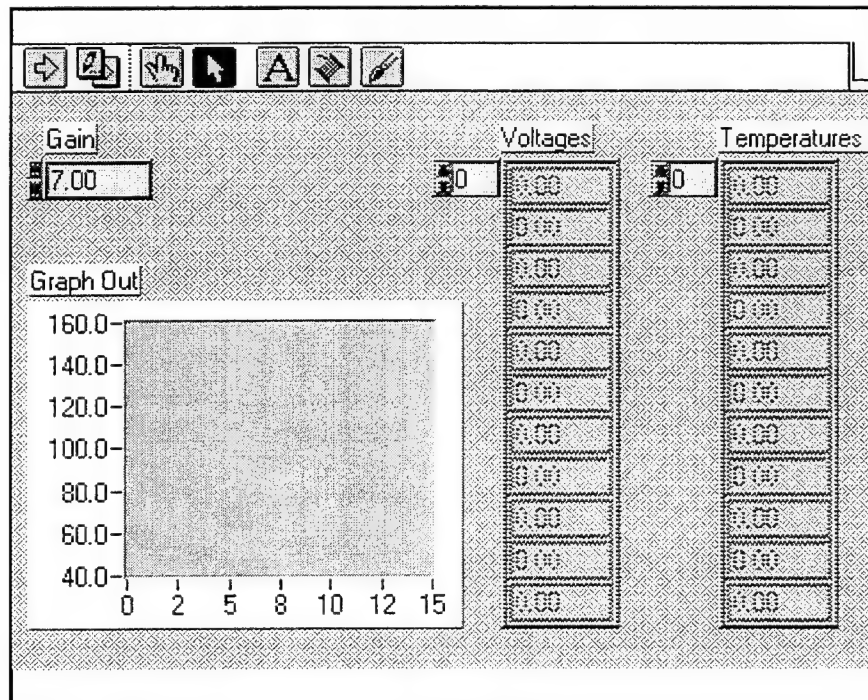


Figure 31: Front Panel for VI *TSWEET2*.

Celsius, where a conversion from voltage to temperature algorithm is employed, then the readings are displayed on the front panel. These readings are then processed by the VI *Graph Array*, where the data is plotted. Additional details on *TSWEEP2* and its related subVIs may be found in Appendix A.

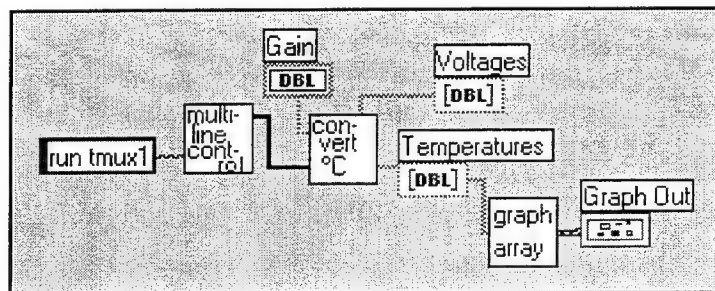
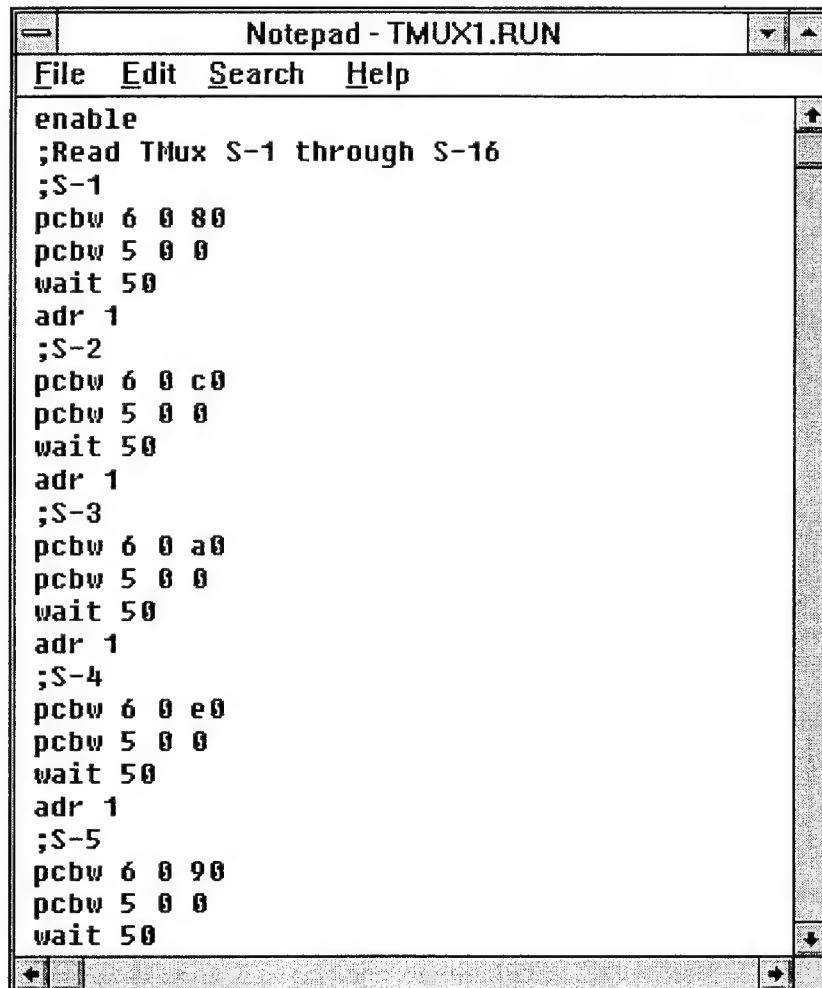


Figure 32: Block Diagram for VI *TSWEEP2*.



```
enable
;Read TMux S-1 through S-16
;S-1
pcbw 6 0 80
pcbw 5 0 0
wait 50
adr 1
;S-2
pcbw 6 0 c0
pcbw 5 0 0
wait 50
adr 1
;S-3
pcbw 6 0 a0
pcbw 5 0 0
wait 50
adr 1
;S-4
pcbw 6 0 e0
pcbw 5 0 0
wait 50
adr 1
;S-5
pcbw 6 0 90
pcbw 5 0 0
wait 50
```

Figure 33: Segment of *Notepad* File TMUX1.

C. EPS INTERFACE

1. Power Supply Driver

The VI *GPIBTEST* was created to control the HP6653A Power Supply from the simulator's PC. The front panel for *GPIBTEST* is shown in Figure 34. For the VI to operate, the power button shown in the upper left portion of panel must be in the "ON" position. The power supply is set to a specified voltage and current by user manipulation of the two controls in the upper right quadrant of the panel. The GPIB address for the HP6653A is "6", and the voltage and current measured at the power supply are displayed in the labeled box and on the meter displays. Further information on the operation of the HP6653A may be found in Reference 13.

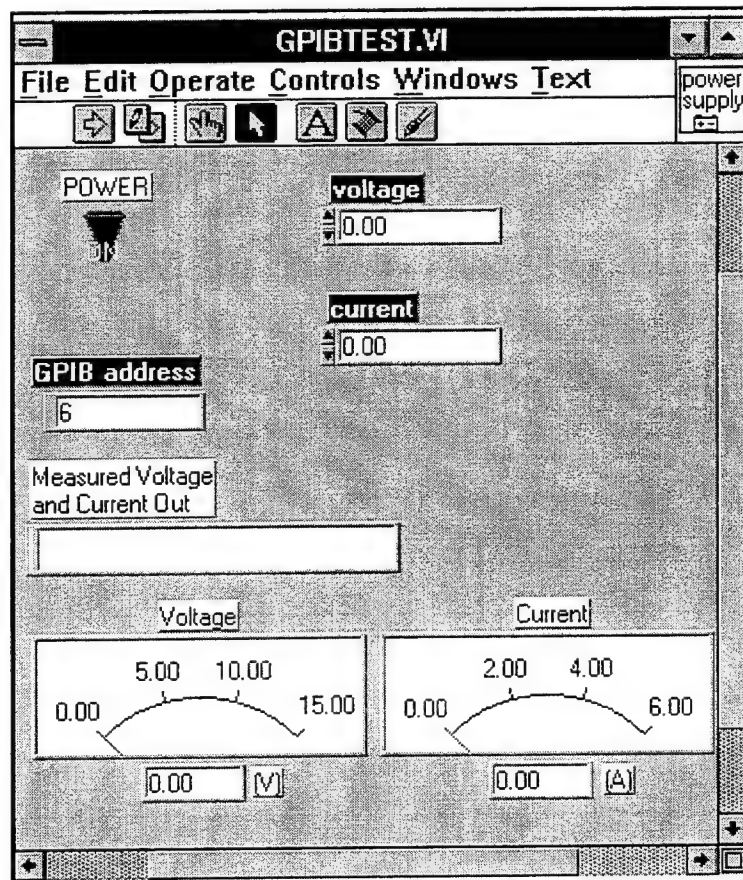


Figure 34: Front Panel for VI *GPIBTEST*.

Alternatively, voltage and current measurements can be sent to the power supply through a user file, such as a compilation of best to worst case scenarios of percent eclipse data. This is accomplished by connecting the data to the “voltage” and “current” connectors displayed in *GPIBTEST*’s icon and connector display (see Figure 35). Block diagrams for *GPIBTEST* are presented in Appendix B.

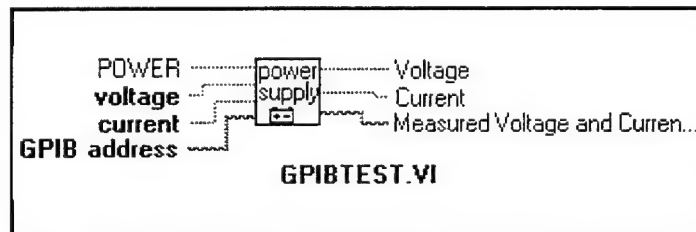


Figure 35: *Icon and Connectors for VI GPIBTEST.*

2. Load Driver

To test the operation of *GPIBTEST*, an electronic load was connected to the power supply. The VI used to control the HP6060A Electronic Load was *GPIBLOAD*, which is shown as an icon and connectors in Figure 36 and as a front panel in Figure 37. As illustrated in the figure, the VI has three modes of operation: constant current (CC), constant resistance (CR), and constant voltage (CV). Only the CC mode has been utilized in testing thus far. As seen in the figure, the GPIB address is indicated as “5”, and voltage and current readings are easily obtained. Block diagrams for this VI are located in Appendix B, and further information on the operation of the HP6060A may be found in Reference 14.

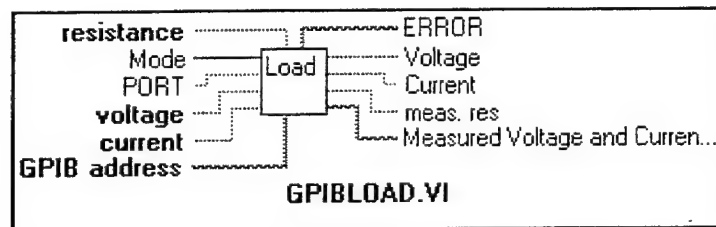


Figure 36: Icon and Connectors for VI GPIBLOAD.

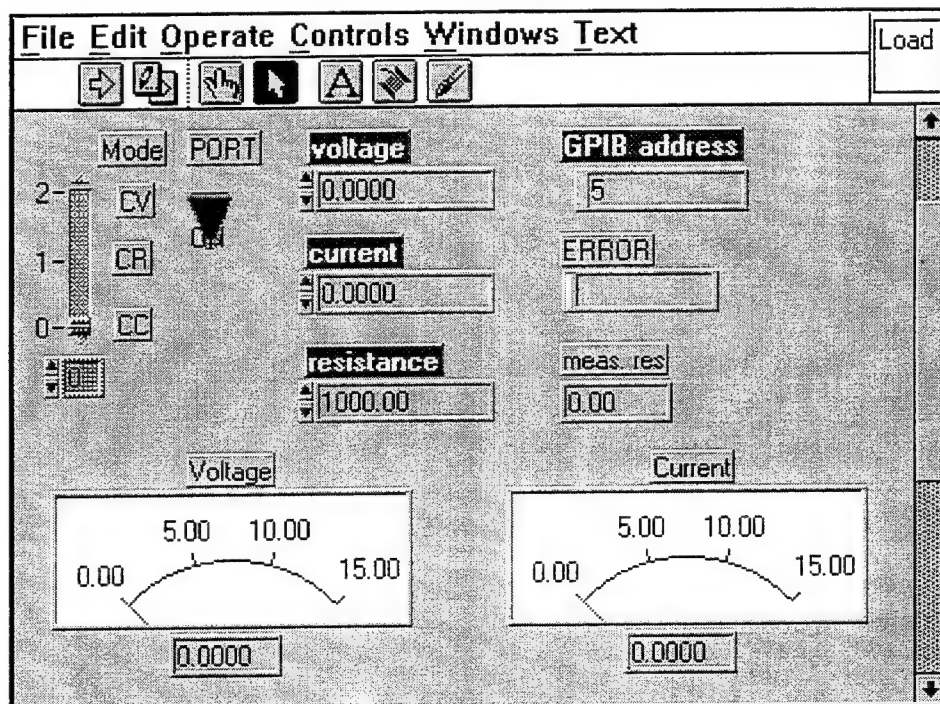


Figure 37: Front Panel of VI GPIBLOAD.

In order to run *GPIBLOAD* to provide an electronic load and *GPIBTEST* to observe the resulting measurements, a third VI, *SUPLD*, was created. Figure 38 illustrates how this VI handles the inputs of mode, voltage, and current, and the resulting measurements. For more information on *SUPLD*, consult Appendix B.

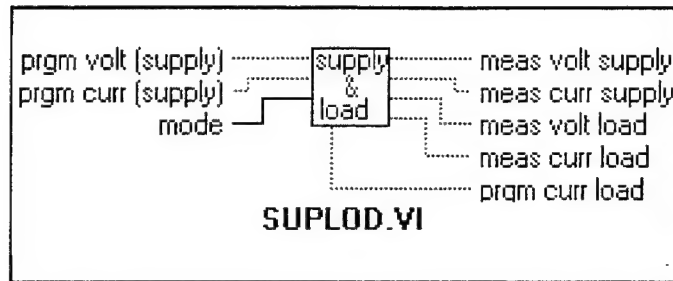


Figure 38: *Icon and Connectors for VI SUPLOD.*

3. Control Panels

Three different VIs were designed to control different aspects of the EPS: battery telemetry, system control and status, and roll rate and attitude. *EPS Battery Telemetry* is shown as an icon and connectors in Figure 39, and is intended to provide the user with a simple to read panel for monitoring the status of the batteries. Development of this VI is still ongoing, and will continue following the arrival and installation of the system's prototype batteries. The front panel for *EPS Battery Telemetry* is pictured in Figure 40, where "Graph Out" is a plot of voltage readings, "Temperature" graphically represents the temperature readings, and "Trickle", "Charge", and "Discharge" buttons indicate to the user the current state of battery operation. Note that data is to be collected on each of 10 cells in both batteries.

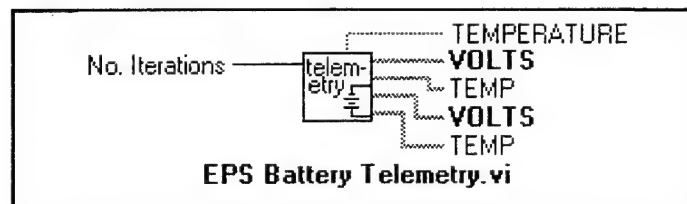


Figure 39: *Icon and Connectors for VI EPS Battery Telemetry.*

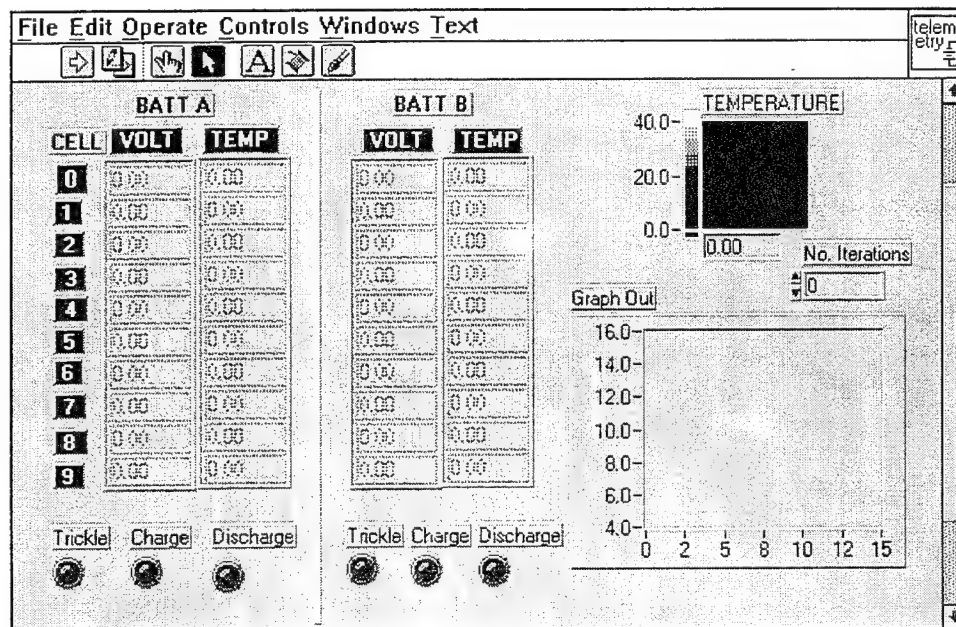


Figure 40: *Front Panel for VI EPS Battery Telemetry.*

Figure 41 presents the VI *EPS Subsystem On/Off* as a front panel. Power to each of the simulator's subsystems is controlled from this panel. To turn a specific subsystem on or off, the user clicks the appropriate switch in the top half of the panel. The paired components have been installed to allow only one to be turned on at a time, and the user is informed that turning off DCS A or B will result in a reset of the entire system. Below the on/off toggle switches are system status buttons, which indicates which subsystems are actually being provided with power. This VI is presented as an icon and connectors in Figure 42, and as a block diagram in Appendix B.

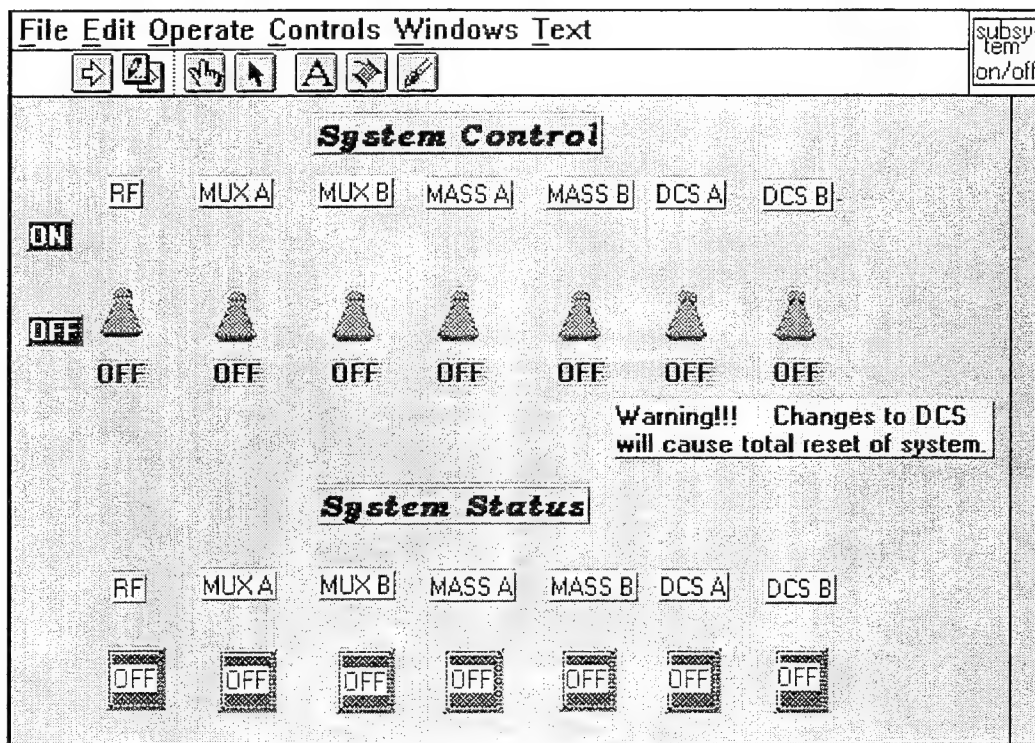


Figure 41: Front Panel for VI EPS Subsystem On/Off.

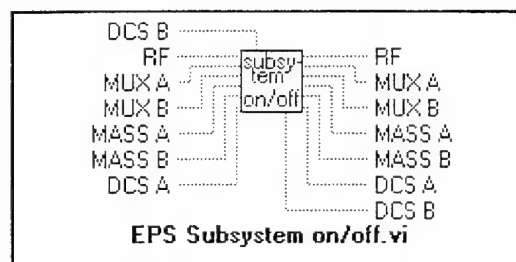


Figure 42: Icon and Connectors for VI EPS Subsystem On/Off.

Eight solar cells on each of eight solar panels will contain a current sensor. These sensors will be distributed in such a manner as to allow the satellite's roll rate and attitude to be calculated. The VI *EPS Roll Rate/Attitude* is being created to receive the sensor data and graphically express it to the user; it is shown as a front panel in Figure 43, as an icon and connector in Figure 44, and can be seen as a block diagram in Appendix B. As with some of the software components discussed above, final design implementation of *EPS Roll Rate/Attitude* is being delayed pending the arrival and installation of the prototype batteries.

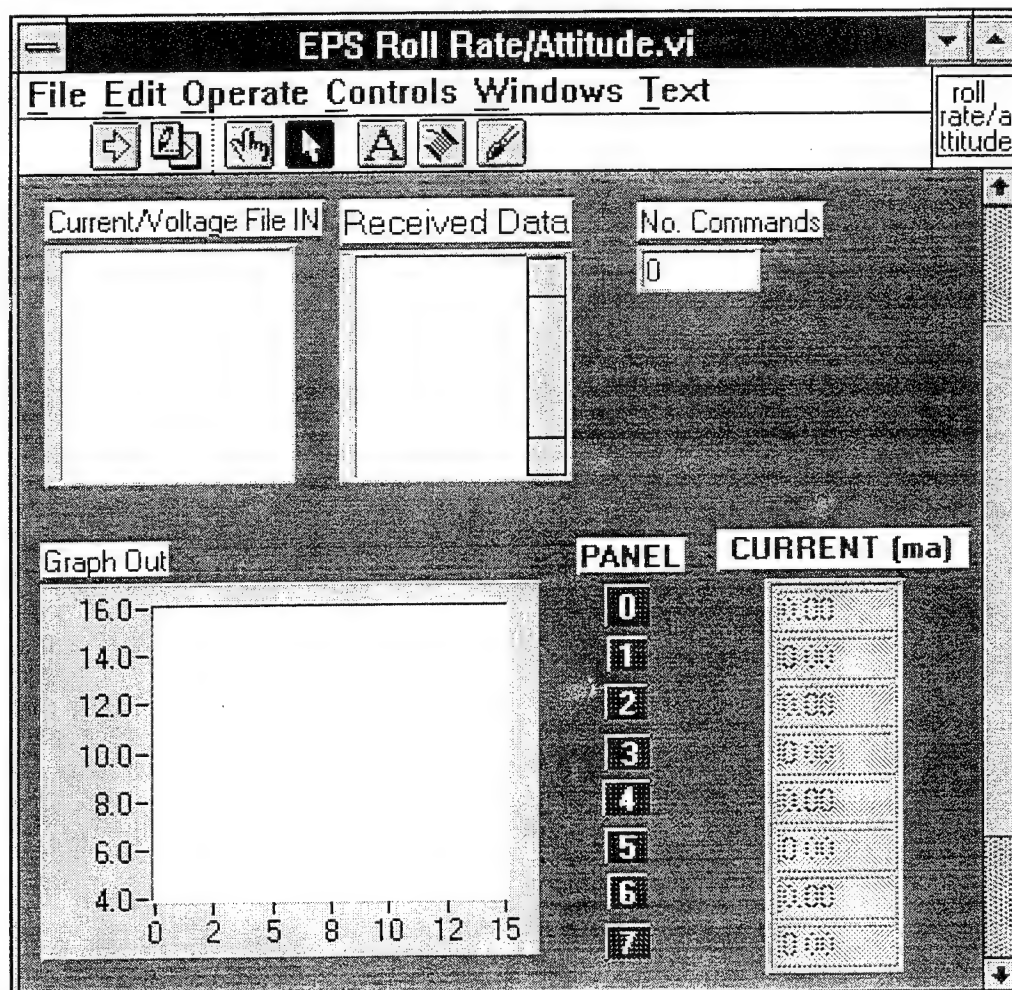


Figure 43: *Front Panel of VI EPS Roll Rate/Attitude.*

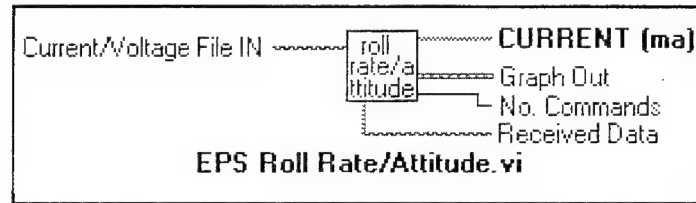


Figure 44: *Icon and Connectors for VI EPS Roll Rate/Attitude.*

The VI *IV File Solar Simulator* uses a current and voltage (IV) file to retrieve data and provide the user a platform upon which to analyze the operation of the batteries. The IV file can contain anticipated or measured current and voltage values reported by the batteries or the solar panels during one complete orbit. During eclipse, power is provided from the batteries, and when the spacecraft is in sunlight the solar panels power the system. Based on the information in these files, numerous orbits can be simulated to see how the batteries respond. As shown in Figure 45, the user can enter the period of orbit in minutes and seconds. The value of “Time Weighting ...” indicates the factor by which time has been accelerated or decelerated; a value of two allows the user to simulate the orbit in half the original time. Percent of completed orbit is also monitored during the simulation, by means of the simulated analog gauge in the upper right portion of the panel. Final design implementation of this VI component is also awaiting the arrival and installation of the prototype batteries. The block diagram for *IV File Solar Simulator* may be found in Appendix B.

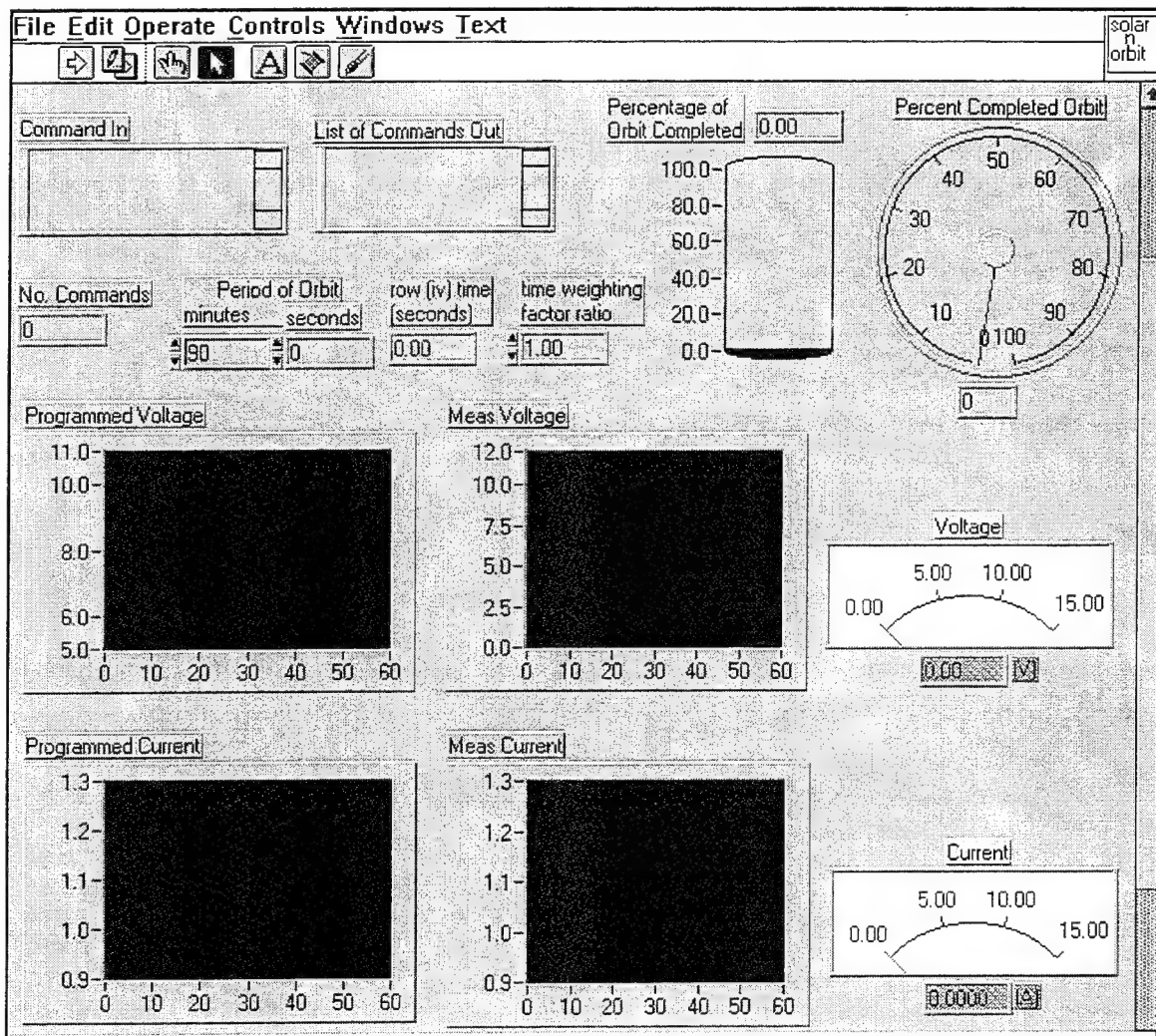


Figure 45: Front Panel for VI IV File Solar Simulator.

V. CONCLUSIONS AND RECOMMENDATIONS

This thesis has described the development of the PANSAT simulator's Digital Control Subsystem (DCS) and Electrical Power Subsystem (EPS) interfaces. Where practical and informative, the discussion has included references to and descriptions of various components of both PANSAT and the simulator, both of which are still in development. The specific laboratory work herein described was concerned exclusively with the development of interface controls for the simulator's DCS and EPS Subsystems.

LabVIEW has been demonstrated to be an easy-to-use and effective programming tool in the development of PC based interfaces. Through the use of Virtual Instruments (VIs), the user is presented with easy-to-use digital readouts and graphical representations of the measured performance of simulator components. Additionally, VIs are useful in controlling peripheral devices that simulate as yet incomplete or uninstalled components. In ongoing testing of the simulator's TMUX, for instance, the interfaces described in Chapter IV have provided engineers with an efficient tool with which to gather and interpret operational data.

There are several specific objectives to be achieved in further research. First, switch configurations and addresses of all elements within the subsystems controlled through the Peripheral Control Bus (PCB) must be identified, collected and organized. Those addresses already identified are contained in Appendix C. Second, the STAR Control Board currently used to simulate microprocessor operations must be replaced with the PANSAT Control Board, which will allow an LM12454 A/D converter to perform analog to digital conversions (as in PANSAT) and the integration of the simulator's Communication Subsystem (COMM).

The DCS interfaces and VI displays discussed in this thesis are in place and functioning within the embedded simulator system at the NPS, and they have been useful in evaluating newly designed or installed system components. As new components come

on line and existing components are revised, however, changes to the interfaces and VI displays may be necessary.

The installation of the LM12454 A/D converter, for instance, will necessitate changes in the programming code in the following manner. *Command Response Cases VI* must be modified to handle *ADR* commands to the microprocessor. *Command Response Cases* block diagrams can be seen in Appendix A. Frame '13' of the *block diagram* is the frame to be modified; the *VI ADR Fake* must be deleted. The *array of u8 in*, shown at the top of Frame 13, must first contain an *Array Subset Function* with a starting index of 0 and a width of 1 byte. This byte, representing *ad_channel*, should feed into the *Add Array Elements Function* and then into the *To Hexadecimal Function*. This string is then passed into the *Build Array Function*, following the *ADR* element already in place in Frame 13, and an equal sign must be inserted into the *Build Array Function* following the *ad_channel* element. The *array of u8 in* should pass into another *Array Subset Function* with a starting index of 1 and a width of 2 bytes. This should be followed by the *VI Change 2([u8]) to u16*, fed into the same set of functions described above for the *ad_channel* element, and added to the *Build Array Function*.

The PANSAT simulator, when complete, will present an ideal tool with which to continue development of the operational PANSAT system. For the present, the continuing design and construction of the simulator itself present graduate students with a unique educational experience.

APPENDIX A. SIMULATOR DIGITAL CONTROL SUBSYSTEM (DCS) INTERFACE SOFTWARE CODE

Virtual instrument (VI) *Multi Line Control* handles all commands sent to the DCS. Figure 46 shows supporting VIs (*subVIs*) for VI *Multi Line Control*. Appendix A, Section A, contains the icon and connectors and block diagrams for the VIs shown in Figure 46. A block diagram is the code for a program in the graphic G language, as discussed in Chapter IV. A brief description of each *diagram* is included. Section B of Appendix A contains *block diagrams* of *TSWEEP2 VI* and its supporting VIs.

A. MULTI LINE CONTROL

VI *Multi Line Control* is supported by a number of *subVIs*. Each VI developed to support *Multi Line Control* is briefly discussed below. Only VIs specifically built for the DCS interface are discussed and shown below starting from the second row from the bottom of Figure 46. The bottom row of Figure 46 contain VIs that initiate or activate the General Purpose Interface Bus (GPIB) and Serial Port Communications. The *General Error Handling VI* is also shown. For further information on these VIs developed by National Instruments see References 9, 12, 15, 16.

1. Change u16 to 2([u8]) VI

Starting on the second row from the bottom, left hand side of Figure 46 is the *Change u16 to 2([u8]) VI*. This VI performs the function of swapping the high and low bytes of a two byte number to fit the 80186 architecture required before it is sent to the microprocessor. Figure 47 is the *icon and connectors* for *Change u16 to 2([u8]) VI*.

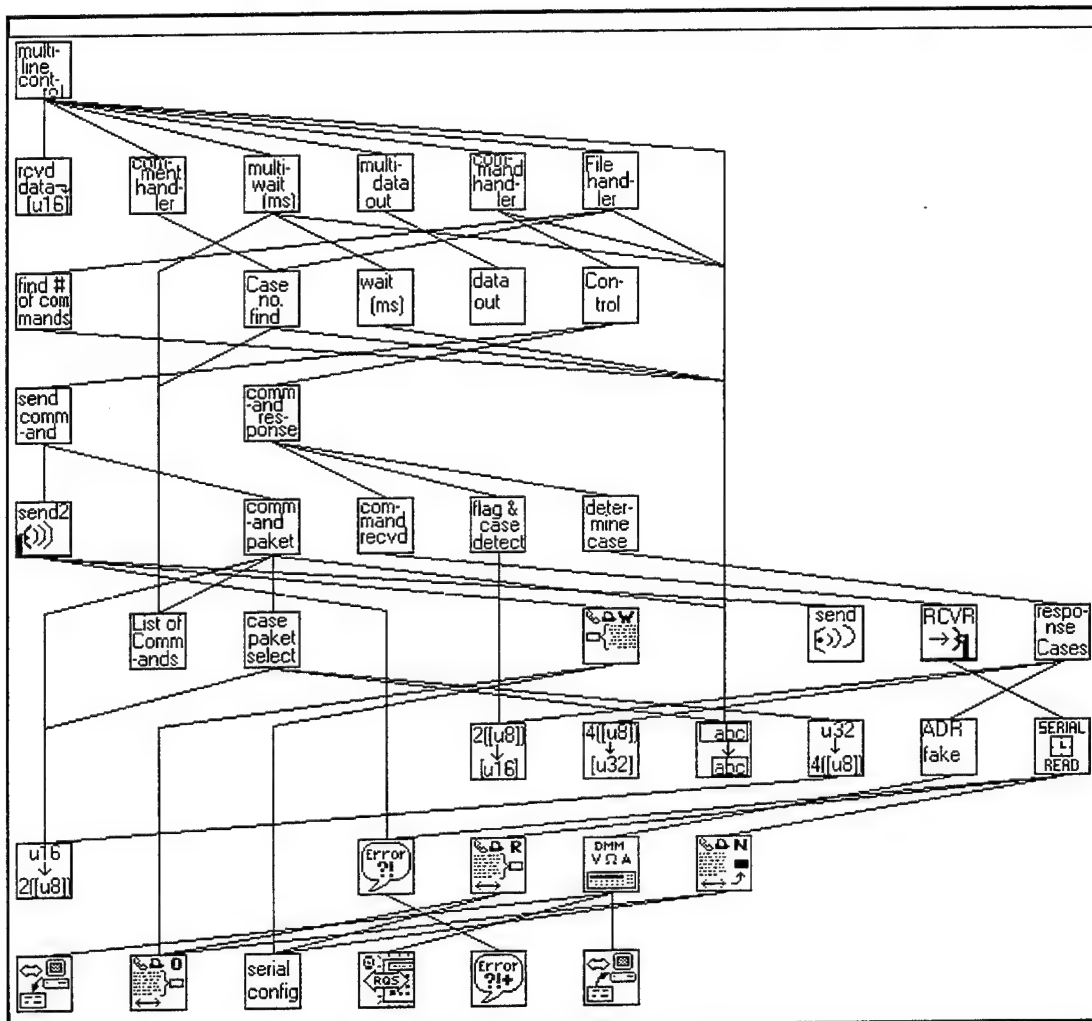


Figure 46: Supporting VIs for VI *Multi Line Control*.

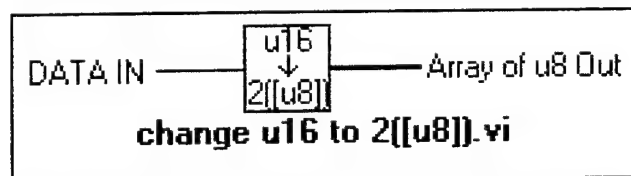


Figure 47: Icon and Connectors for VI *Change u16 to 2[[u8]].vi*

Figure 48 is the *block diagram* for VI *Change u16 to 2([u8])*. The data is received, divided by 256, then sent to an array building function with the remainder of the operation being the first element and the quotient the second.

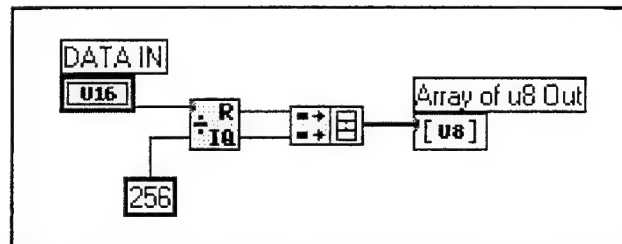


Figure 48: Block Diagram for VI *Change u16 to 2([u8])*.

2. Simple Error Handler, Serial Port Read and Bytes at Serial Port VIs

The *icon and connectors* for VI *Simple Error Handler* is shown in Figure 49 with an explanation of the VI's function. *Simple Error Handler VI*, *Serial Port Read VI* and *Bytes at Serial Port VI*, shown on the same line in Figure 46, were developed by National Instruments. *Serial Port Read VI* and *Bytes at Serial Port VI* were discussed briefly in Chapter VI. For further reading on *Simple Error Handler*, *Serial Port Read* and *Bytes at Serial Port VIs* see the References 9, 12, 15, and 16.

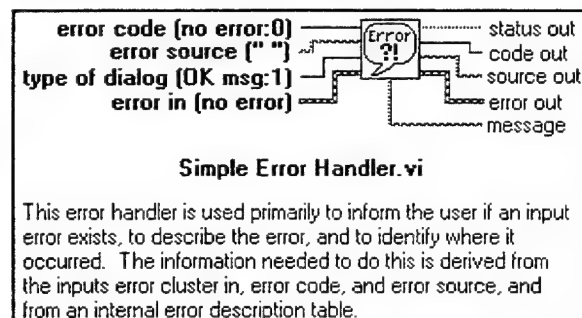


Figure 49: Icon and Connectors for VI *Simple Error Handler*.

3. HP3478A VI

The *icon and connectors* for VI *HP3478A* is shown in Figure 50. The HP3478A VI was developed by National Instruments and is used by the DCS interface software to control the HP3478A multimeter from the PC. The multimeter is used to perform analog to digital conversion as discussed previously in Chapter IV.

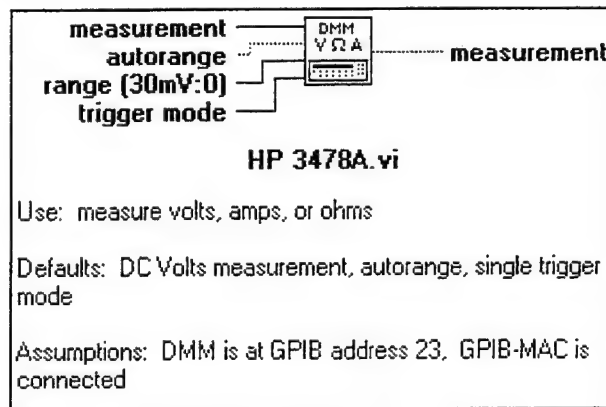


Figure 50: *Icon and Connectors* for VI *HP3478A*.

Figure 51 is the 'false' case, sequence '0' of the block diagram for VI *HP3478A*. Sequence '0' is the first action executed when this VI is activated. This sequence

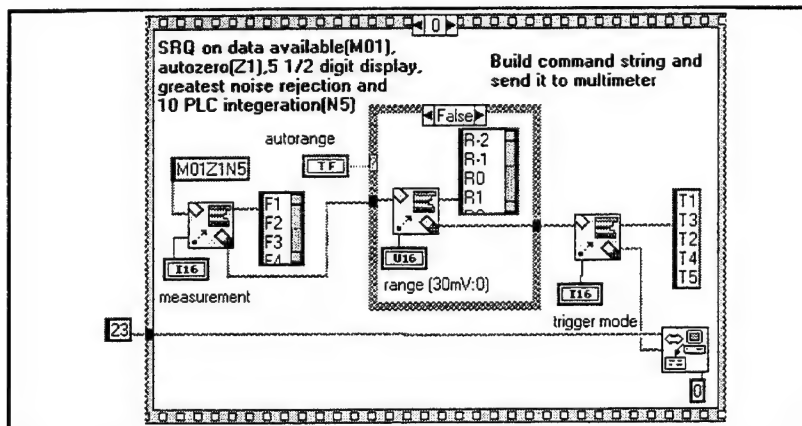


Figure 51: *Block Diagram* for VI *HP3478A* - Sequence '0', 'False' Case.

configures the multimeter, preparing it for taking measurements. The 'false' case occurs when the user chooses not to select the auto ranging mode.

Figure 52 is the 'true' case, sequence '0' of the *block diagram* for VI HP3478A. The 'true' case is executed when the user chooses the auto ranging mode.

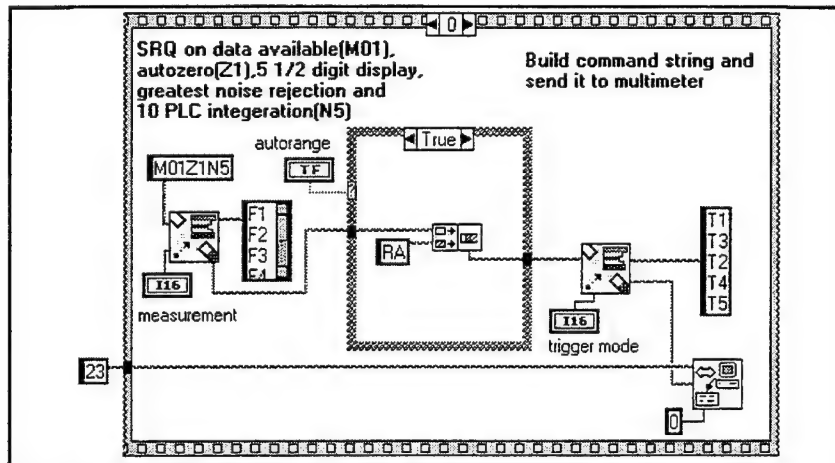


Figure 52: Block Diagram for VI HP3478A - Sequence '0', 'True' Case.

Figure 53 is sequence '1' of the *block diagram* for VI HP3478A. Sequence '1' pauses operations until the multimeter indicates that measurements have been made and are ready for reading.

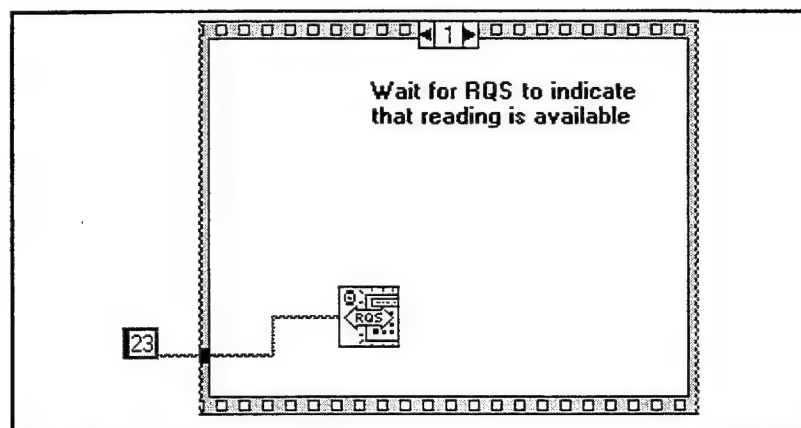


Figure 53: Block Diagram for VI HP3478A - Sequence '1'.

Figure 54 is sequence '2' of the *block diagram* for VI HP3478A. Sequence '2' contains the code that performs the read operation from the multimeter through the GPIB.

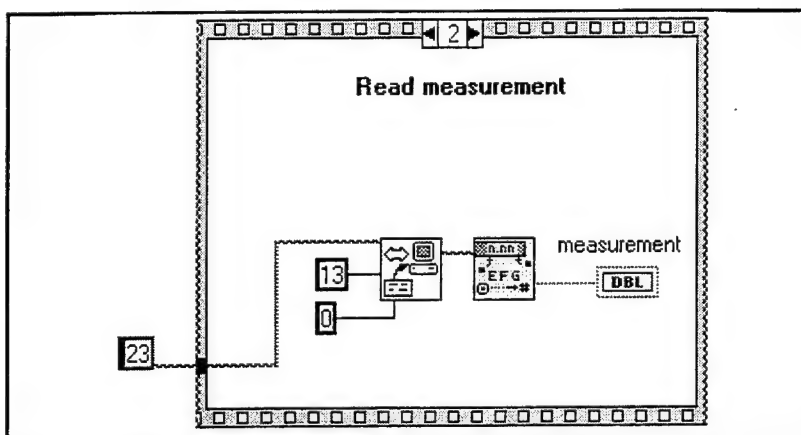


Figure 54: *Block Diagram* for VI HP3478A - Sequence '2'.

Figure 55 is sequence '3' of the *block diagram* or code for VI HP3478A. Sequence '3' clears the multimeter, preparing it for future use.

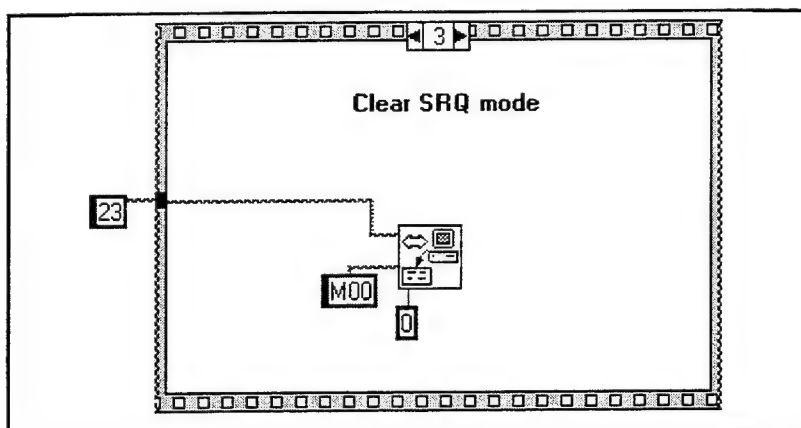


Figure 55: *Block Diagram* for VI HP3478A - Sequence '3'.

For further reading on the operation and remote programming of the HP3478A Multimeter, see Reference 17.

4. Change 2([u8]) to u16 VI

Figure 56 shows the *icon and connectors* for VI *Change 2([u8]) to u16*. This VI receives from the microprocessor two unsigned bytes of data that are in the 80186 format. The high and low bytes need to be swapped and combined to be formatted into the form that the user is used to.

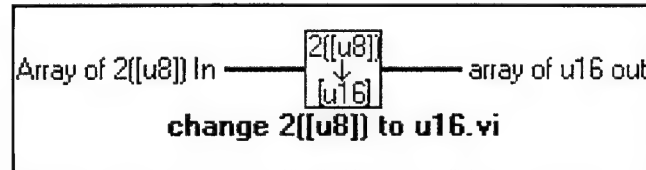


Figure 56: *Icon and Connectors* for VI *Change 2([u8]) to u16*.

Figure 57 is the *block diagram* or code that performs the VI *Change 2([u8]) to u16* function.

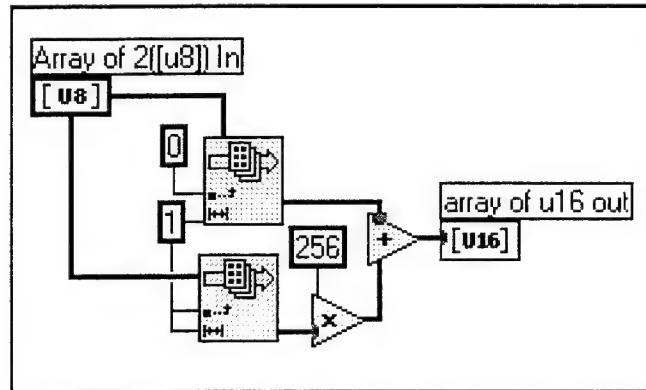


Figure 57: *Block Diagram* for VI *Change 2([u8]) to u16*.

5. Change 4([u8]) to u32 VI

Figure 58 shows the *icon and connectors* for VI *Change 4([u8]) to u32*. This VI receives from the microprocessor four unsigned bytes of data that are in the 80186 format. The high and low bytes need to be swapped and combined to read out

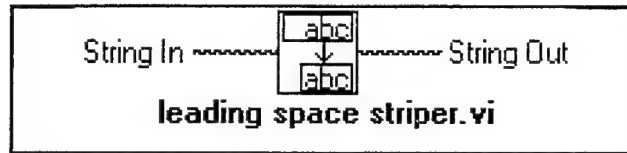


Figure 60: *Icon and Connectors for VI Leading Space Stripper.*

The algorithm used to accomplish the *Leading Space Stripper VI* function is shown in figure 61. Figure 61 is the *block diagram* for VI *Leading Space Stripper*.

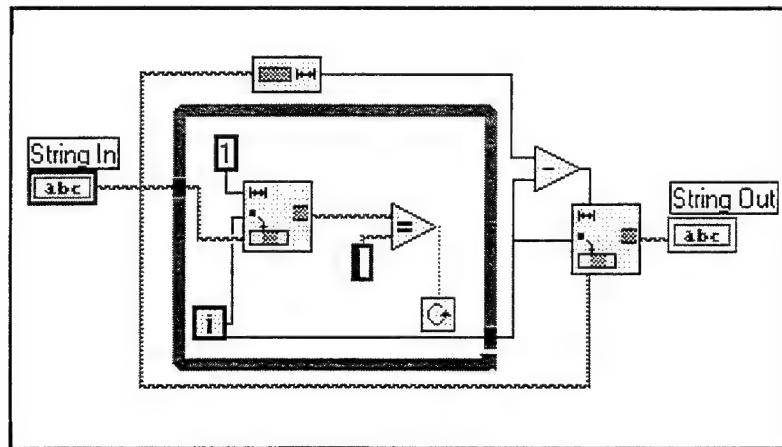


Figure 61: *Block Diagram for VI Leading Space Stripper.*

7. Change u32 to 4([u8]) VI

Figure 62 shows the *icon and connectors* for VI *Change u32 to 4([u8])*. VI *Change u32 to 4([u8])* transforms a 32 bit number into the 80186 architecture format. This prepares the data for handling by the microprocessor.

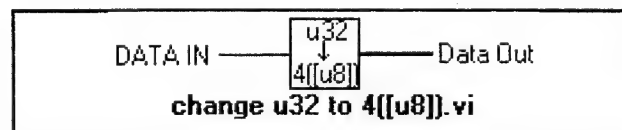


Figure 62: *Icon and Connectors for VI Change u32 to 4([u8]).*

The algorithm to accomplish this is shown in Figure 63, the *block diagram* for VI *Change u32 to 4([u8])*. VI *Change u16 to 2([u8])* is shown in Figure 63 as a *subVI* of VI *Change u32 to 4([u8])*.

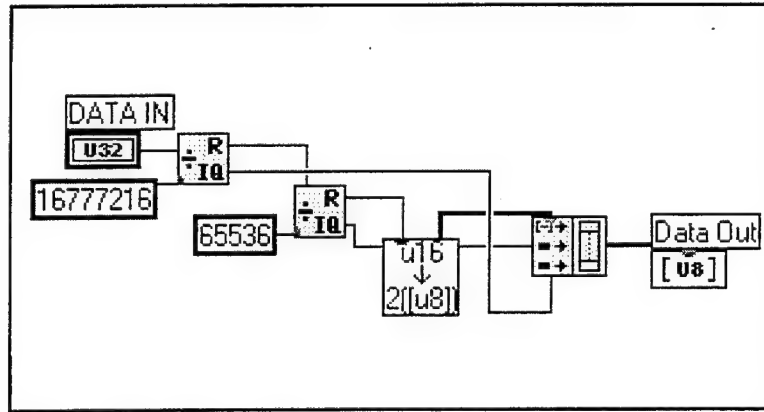


Figure 63: *Block Diagram for VI Change u32 to 4([u8]).*

8. ADR Fake VI

Figure 64 shows the *icon and connectors* for VI *ADR Fake*. VI *ADR Fake* was previously discussed in Chapter IX under the recommendations section. VI *ADR Fake* uses the General Purpose Interface Bus (GPIB) to control the HP3478A Multimeter.

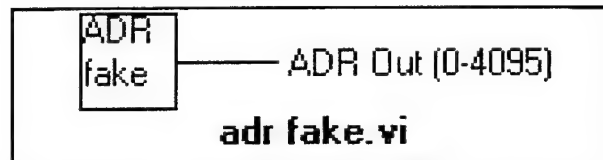


Figure 64: *Icon and Connectors for VI ADR Fake.*

Figure 65 shows the *block diagram* for VI *ADR Fake*. VI *HP3478*, already discussed above, is a *subVI* of *ADR Fake VI*. *ADR Fake VI* converts analog volt readings to digital.

9. List of Commands VI

Figure 68 is the *icon and connectors* for VI *List of Commands*. *List of Commands VI* lists all commands currently served by the simulator's Digital Control Subsystem

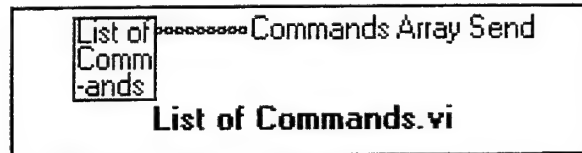


Figure 68: *Icon and Connectors* for VI *List of Commands*.

(DCS) interface software. Commands that the user inputs are compared to this list to determine if the command is valid. If the command is in the list, a 12 bit hexadecimal number is assigned to the command that corresponds to its position in the list. Figure 69 shows the *block diagram* for VI *List of Commands*. *List of Commands VI* is a list stored as an array. New commands can be easily added to the list.

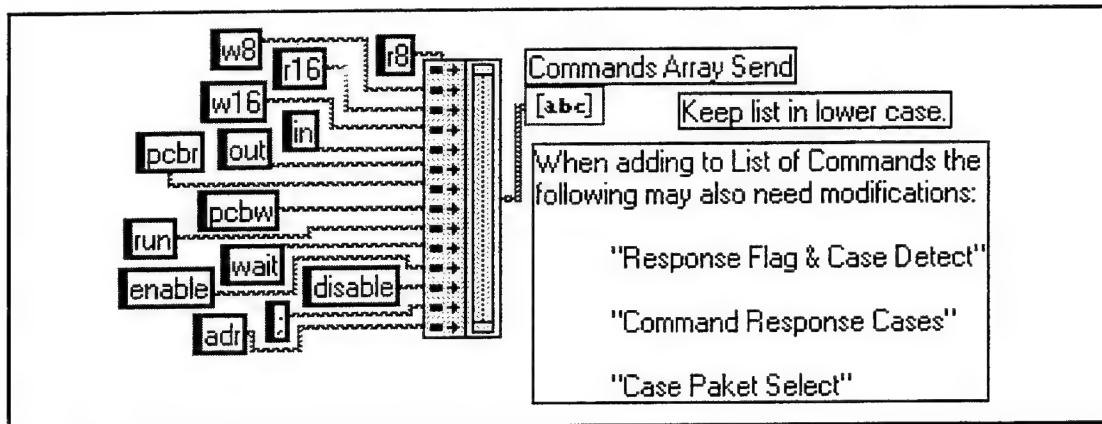


Figure 69: *Block Diagram* for VI *List of Commands*.

10. Case Paket Select VI

Figure 70 is the *icon and connectors* for VI *Case Paket Select*. *Case Paket Select* takes a command string input by the user and transforms the string into the format that is

acceptable by the microprocessor. Each of the 14 commands shown in Figure 69 of the *List of Commands VI block diagram* has a corresponding case shown below in Figures

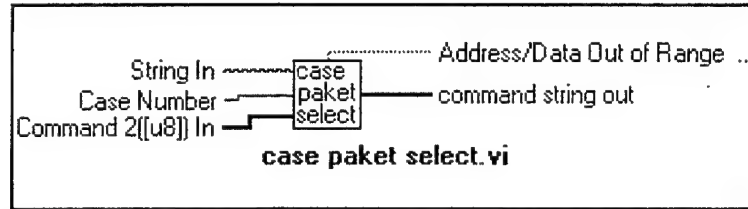


Figure 70: Icon and Connectors for VI Case Packet Select.

71 and 73 through 81. Case numbers are located at the top of each of the boxes shown in Figures 71 and 73 through 81. Figure 71 is the algorithm used to transform the *Memory Read (8-bit) Command* into a format that is acceptable for processing by the

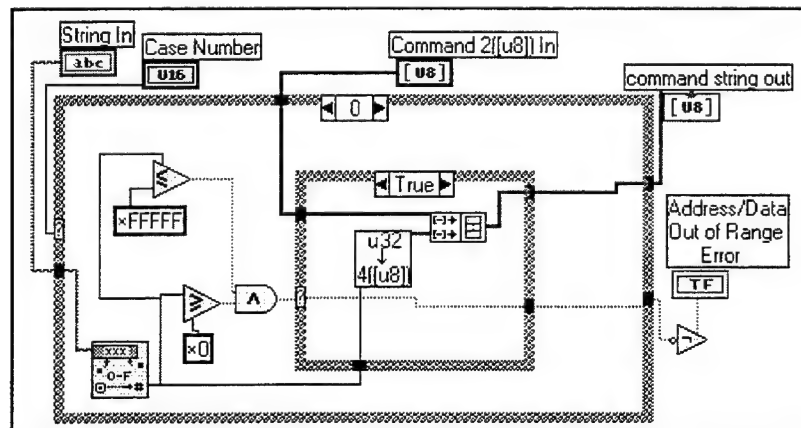


Figure 71: Block Diagram for VI Case Packet Select - Case Number '0'.

microprocessor. The algorithm combines into an array the hexadecimal equivalent for the *Memory Read (8-bit) Command* and its 32 bit CPU address. This array string is output to the indicator labeled *command string out*. The hexadecimal equivalent for *Memory Read (8-bit)* is '0', corresponding with the case number shown in the *block diagram* of Figure 71. All Commands that follow, Figures 73 through 81, have a hexadecimal command

number that corresponds to it's case number. Additionally, the CPU address range is checked to ensure that it is within the specified range. If the address falls within this range, the case is considered true and the code shown in the 'true' box of Figure 71 is executed. If the address falls outside the predefined range, the case is considered false and the code shown in the 'false' box of Figure 72 is executed. For Figures 73 through 81, this code for the false case applies whenever input data is out of the specified range. If an address falls outside the specified range, operations cease and an input error is indicated to the user. If the address is within range, the address is passed through the VI *Change u32 to 4([u8])*. This transforms the CPU address into the necessary format (80186 architecture) for handling by the microprocessor.

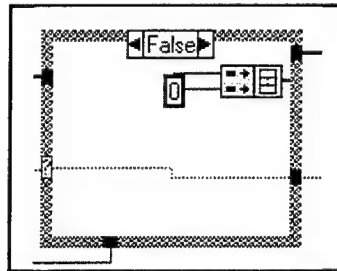


Figure 72: 'False' Case for Frames '0' through '13'.

Figure 73 is the *block diagram* used to prepare the *Memory Write (8-bit) Command* string for processing by the microprocessor. It's case number is '1', which corresponds to the *Memory Write (8-bit) Command's* hexadecimal equivalent. The CPU address range is checked and then passed through the *Change u32 to 4([u8]) VI* as it was in the proceeding *Memory Read Command*. In addition, the *Memory Write Command* contains the 8-bit data input which first has it's range checked for validity. Then the 8-bit data is added to the output array indicated as *command string out*.

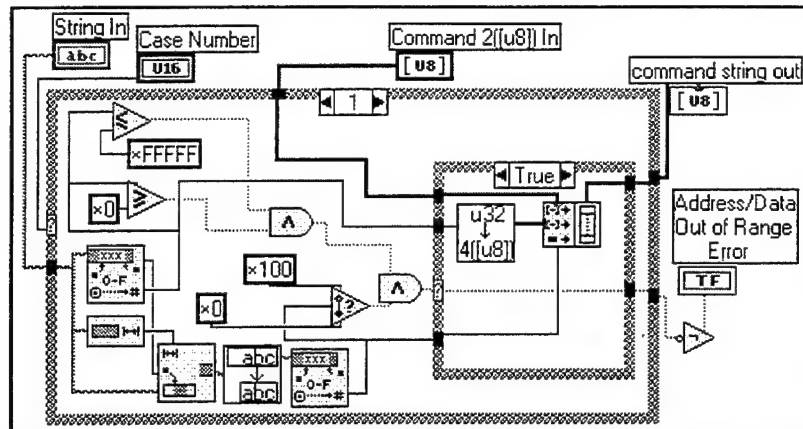


Figure 73: Block Diagram for VI Case Paket Select - Case Number '1'.

Figure 74 is the *block diagram* used to prepare the *Memory Read (16-bit) Command* string for processing by the microprocessor. It's case number is '2', which corresponds to the *Memory Read (16-bit) Command's* hexadecimal equivalent. The CPU address range is checked and then passed through the *Change u32 to 4([u8]) VI*. The address is combined in an array with the command's hexadecimal number equivalent and then indicated as *command string out*.

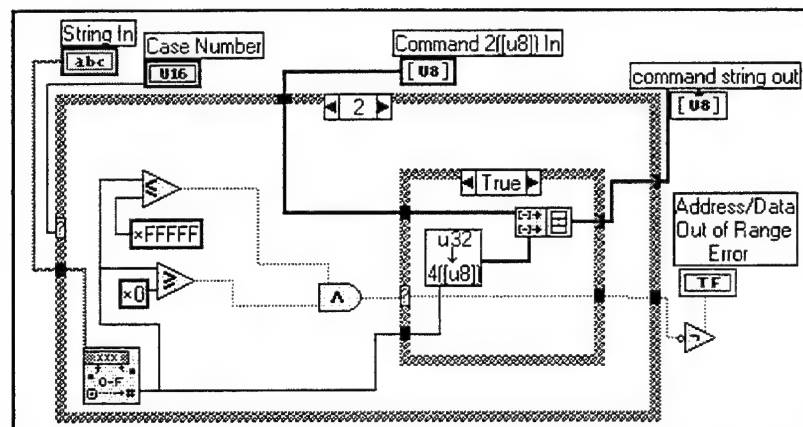


Figure 74: Block Diagram for VI Case Paket Select - Case Number '2'.

number equivalent is combined in an array with the 80186 formatted CPU I/O port number. The output array is indicated as *command string out*.

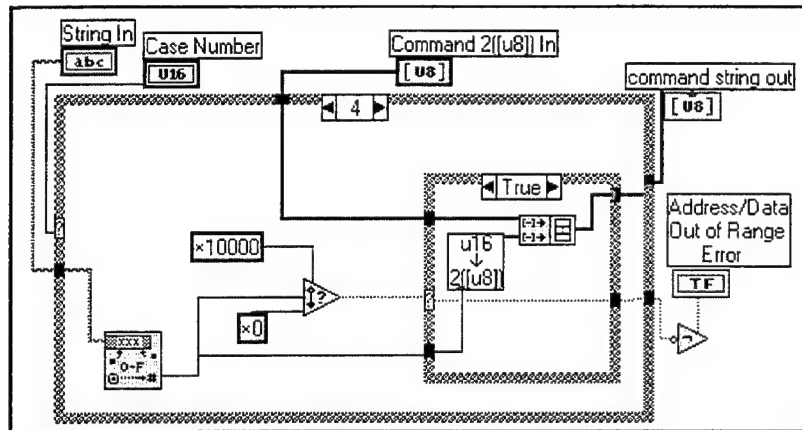


Figure 76: Block Diagram for VI Case Paket Select - Case Number '4'.

Figure 77 is the *block diagram* used to prepare the *I/O Port Write Command* string for processing by the microprocessor. It's case number is '5', which corresponds to the *I/O Port Write Command's* hexadecimal equivalent. The CPU I/O port number is passed through the *Change u16 to 2([u8]) VI* as it was for the *I/O Port Read Command*. The *I/O Port Write Command's* hexadecimal number equivalent is combined in an array with the 80186 formatted CPU I/O port number. In addition, the *I/O Port Write Command* combines the 8-bit data input into the output array that is indicated as *command string out*.

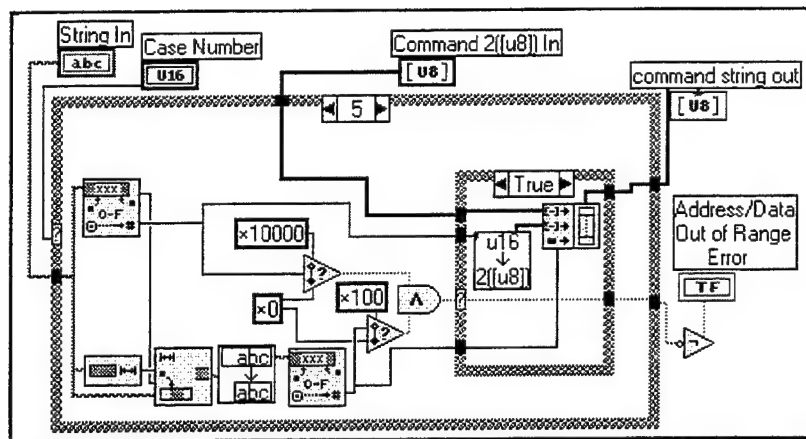


Figure 77: Block Diagram for VI Case Paket Select - Case Number '5'.

Figure 78 is the *block diagram* used to prepare the *PCB Read Command* string for processing by the microprocessor. It's case number is '6' corresponding to the *PCB Read Command's* hexadecimal equivalent. The Peripheral Control Bus (PCB) device address and PCB device sub address ranges are verified. The PCB device address and sub address are combined with the *PCB Read Command's* hexadecimal number equivalent into an array whose output is indicated as *command string out*.

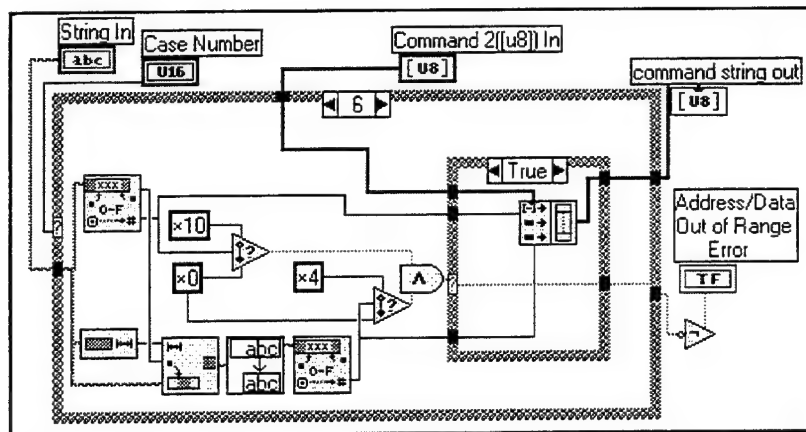


Figure 78: Block Diagram for VI Case Paket Select - Case Number '6'.

Figure 79 is the *block diagram* used to prepare the *PCB Write Command* string for processing by the microprocessor. It's case number is '7' corresponding to the *PCB Write Command's* hexadecimal equivalent. The Peripheral Control Bus (PCB) device address, PCB device sub address, and 8-bit input data ranges are verified. The PCB device address, sub address, and 8-bit input data are combined with the *PCB Read Command's* hexadecimal number equivalent into an array whose output is indicated as *command string out*.

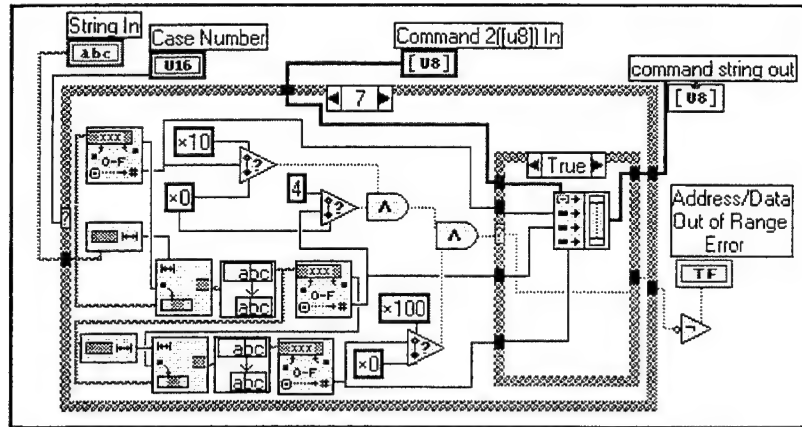


Figure 79: Block Diagram for VI Case Paket Select - Case Number '7'.

Figure 80 is the *block diagram* containing the code for the *Run Command*. It's case number is '8' corresponding to the *Run Command's* hexadecimal equivalent. The *Run Command* is not sent directly to the microprocessor. This *block diagram* for the *Run Command* serves more as a place marker for future additions to the list of commands. The *Wait Command's* *block diagram* is similar to the *Run Command's* *block diagram* shown in Figure 80 except that the case number is '9' instead of '8'. The *Enable* and *Disable Commands* are also similar to the *Run Command's* *block diagram* except that their case numbers are '10' and '11', respectively. The *Enable* and *Disable Commands* are handled by the microprocessor, but because the commands of each consist of only their

Commands' hexadecimal number equivalents, the numbers are passed directly through to the *command string out* indicator. *Comment Command* is handled the same as the *Run* and *Wait Commands* with a hexadecimal number equivalent of '12'.

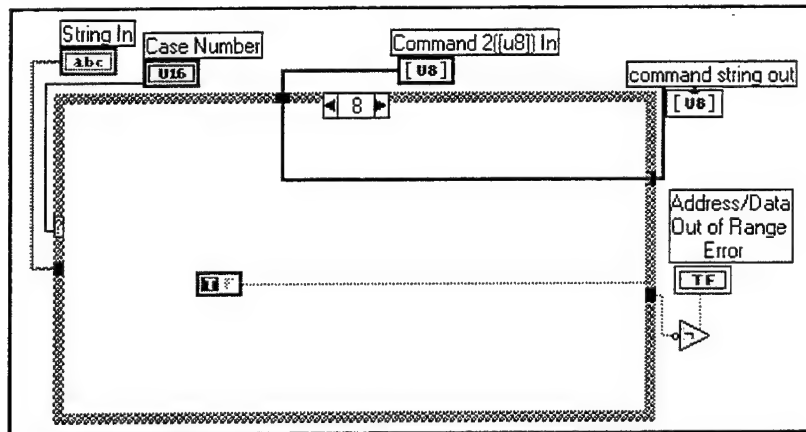


Figure 80: Block Diagram for VI Case Paket Select - Case Number '8'.

Figure 81 is the *block diagram* used to prepare the *Read A/D Converter Command* string for processing by the microprocessor. It's case number is '13' corresponding to the *Read A/D Converter Command's* hexadecimal number equivalent. The A/D converter's channel range is verified. The A/D converter's channel number is combined with the *A/D Converter Command's* hexadecimal number equivalent into an array whose output is indicated as *command string out*. This command is ignored by the microprocessor at this time. Once the STAR Control Board is replaced by the PANSAT Control Board, the microprocessor will handle A/D conversions.

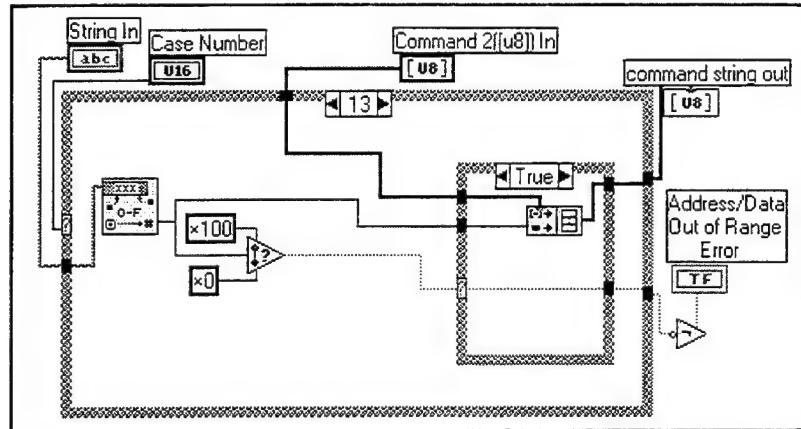


Figure 81: Block Diagram for VI Case Paket Select - Case Number '13'.

11. Serial Port Write VI

Serial Port Write VI was discussed in Chapter VI. *Serial Port Write VI* was developed by National Instruments and is further discussed in References 9, 12, 15, and 16.

12. Send1 VI

Figure 82 is the *icon and connectors* for VI *Send1*. *Send1 VI* creates the *Serial Port Packet* shown in Table 2 of Chapter IV. A packet consist of the *Flag*, *Len1*, *Len2*, *Data Field*, *CRC1* and *CRC2*. This VI consist of six sequences. Sequences '0' through '5' are shown below with a brief description of each.

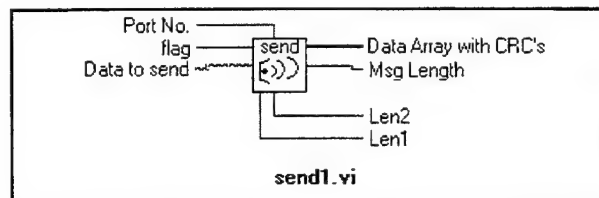


Figure 82: Icon and Connectors for VI *Send1*.

Shown in Figure 83 is sequence '0' of VI *Send1*. The code in sequence '0' takes the input command string and finds its length. The message length is divided by 256 with the quotient being stored in Len2 and the remainder in Len1. These values are passed on.

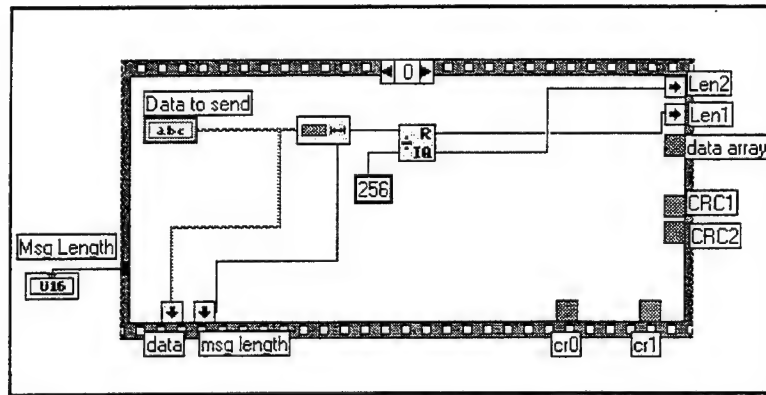


Figure 83: Block Diagram for VI *Send1* - Sequence '0'

Shown in Figure 84 is sequence '1' of VI *Send1*. The code in sequence '1' combines the *Flag* (hexadecimal 7E) with *Len1* and *Len2* in an array builder. This array is then converted into an ASCII string and is then combined with the data string and two zero values that represent the *CRC1* and *CRC2* values until their real values are determined.

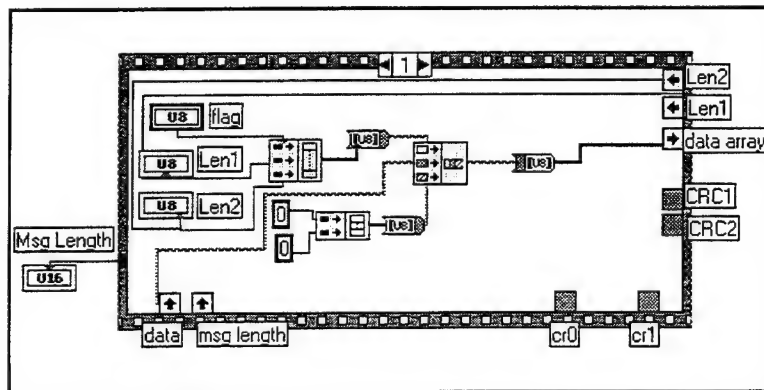


Figure 84: Block Diagram for VI *Send1* - Sequence '1'.

Shown in Figures 85, 86 and 87 are sequences '2', '3' and '4' of VI *Send1*. The code in each of the sequences is determining values *CRC1* and *CRC2* so that cyclic redundancy check (CRC) can be performed when the data is received by the microprocessor.

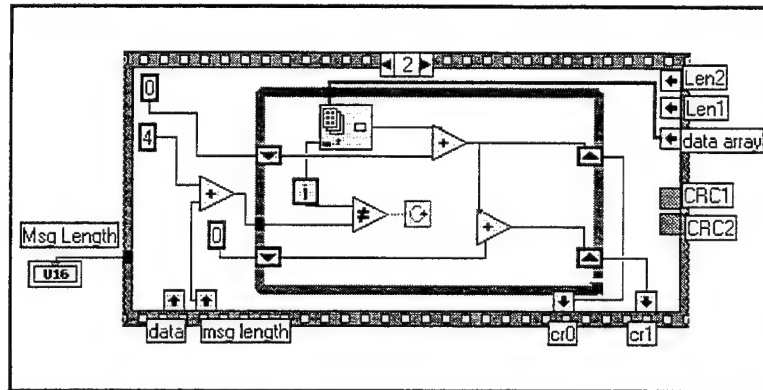


Figure 85: Block Diagram for VI *Send1* - Sequence '2'.

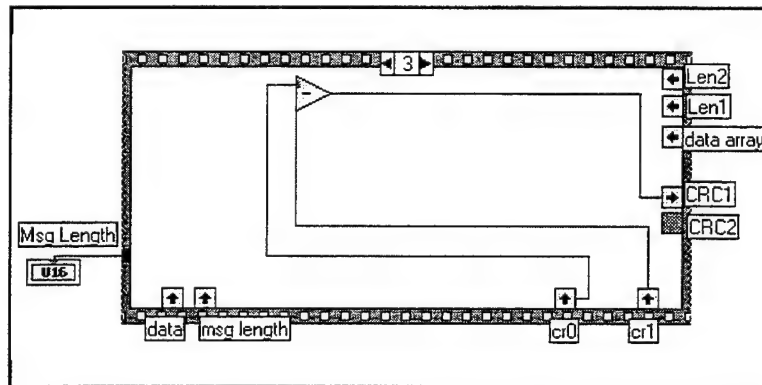


Figure 86: Block Diagram for VI *Send1* - Sequence '3'.

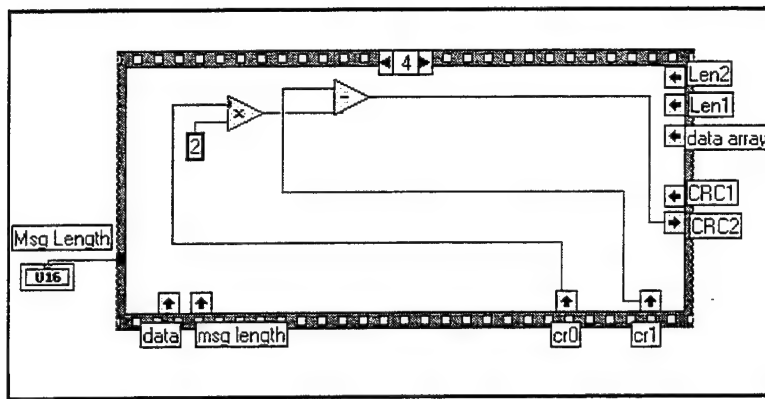


Figure 87: Block Diagram for VI Send1 - Sequence '4'.

Figure 88 shows the code for sequence '5' of VI Send1. This final sequence replaces the zero values that were previously inserted into the *CRC1* and *CRC2* slots of the array with the values that were determined in sequences '2' through '4'. Now the data array is ready to be sent to the microprocessor.

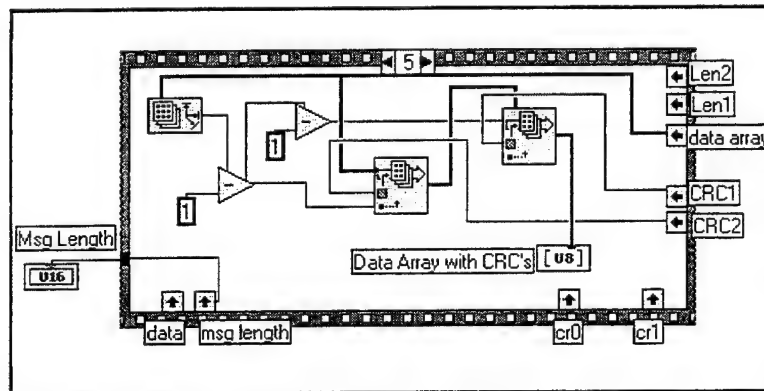


Figure 88: Block Diagram for VI Send1 - Sequence '5'.

13. RCVR VI

Figure 89 is the *icon and connectors* for VI RCVR. RCVR VI processes the string of data that is received from the microprocessor. VI RCVR consists of four sequences described below.

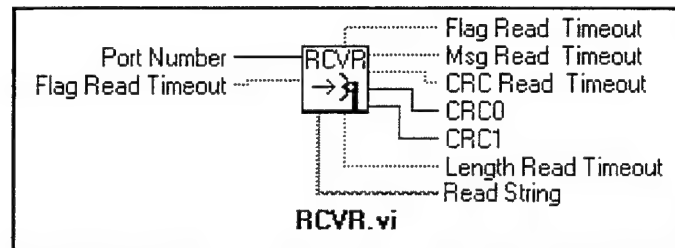


Figure 89: Icon and Connectors for VI RCVR.

Figure 90 shows the *Flag Check* and Sequence '0' portion of the VI RCVR. The *Flag Check* portion is on the left hand side of the figure in the While Loop. The While Loop continues until the *Flag* (hexadecimal 7E) is read from the serial port. One byte is read at a time.

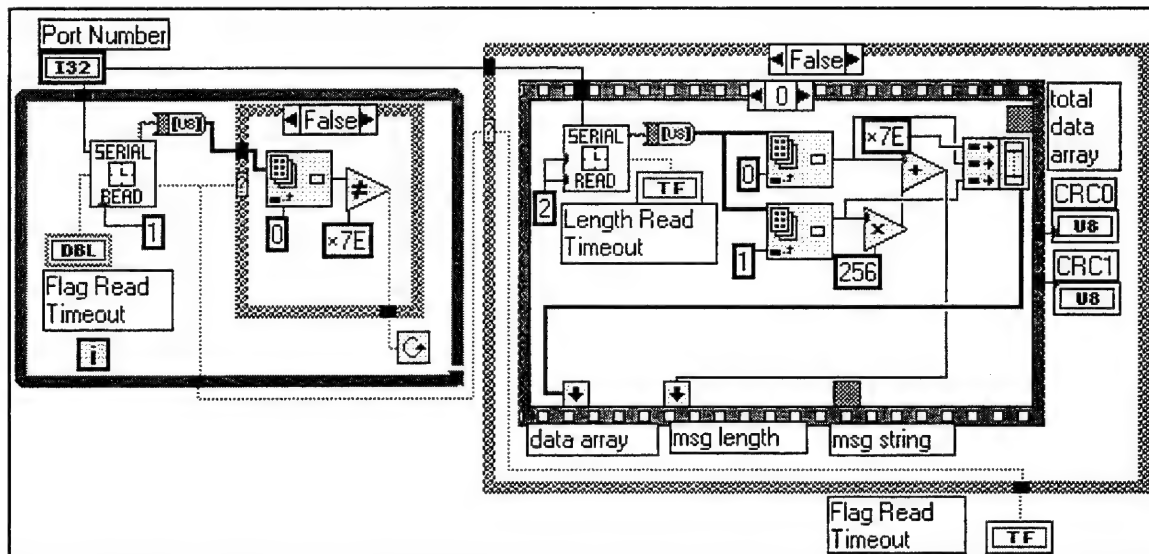


Figure 90: Block Diagram for VI RCVR - Flag Check and Sequence '0'.

Once the *Flag* byte has been read, the data string is passed on to sequence '0' shown in Figure 90. The code in sequence '0' reads the next two bytes of the string to

The code shown in sequence '2' of Figure 93 reads the next two bytes from the serial port - *CRC1* and *CRC2*. All portions of the received data are then recombined to perform the next step - the cyclic redundancy check (CRC).

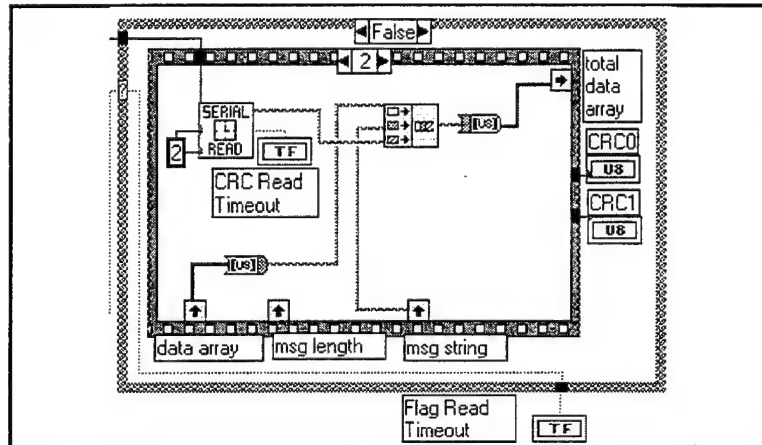


Figure 93: Block Diagram for VI RCVR - Sequence '2'.

Sequence '3' of Figure 94 shows the code for performing the cyclic redundancy check. When calculations are completed, if *CRC0* and *CRC1* are equal to zero then a no error indication for the received string is indicated. If these values are not zero, an error is indicated and retransmission is necessary.

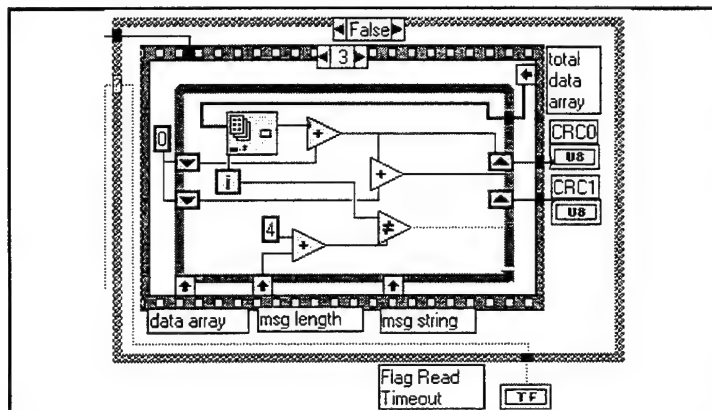


Figure 94: Block Diagram for VI RCVR - Sequence '3'.

14. Command Response Cases VI

Figure 95 shows the *icon and connectors* for VI *Command Response Cases*. This VI is responsible for formatting the command string received from the microprocessor into a format that is easily readable by the user.

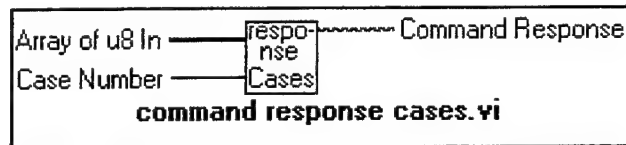


Figure 95: *Icon and Connectors for VI Command Response Cases.*

Figure 96 is the *block diagram* that contains the code for transforming the *Memory Read (8-bit) Command, Case '0'*, into a readable format when it is received back

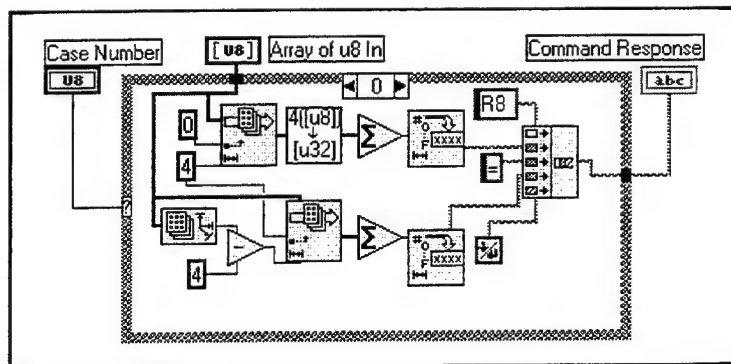


Figure 96: *Block Diagram for VI Command Response Cases - Case '0'.*

from the microprocessor. The 32 bit CPU address is transposed from the 80186 format by the *Change 4([u8]) to [u32] VI*. The 8-bits of data that were requested by the user is delineated by a leading equal sign. The command is read out to the user in an ASCII string format.

Figure 97 contains the *block diagram* for case '1' of VI *Command Response Cases*. Case '1' is the *Memory Write (8-bit) Command*. This command is repeated back

to the user in the same format that it was sent. The 32 bit CPU address is transposed from the 80186 format by the *Change 4([u8]) to [u32] VI*.

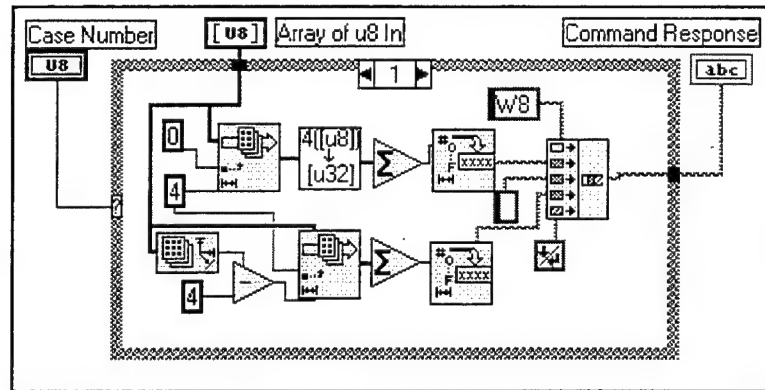


Figure 97: Block Diagram for VI Command Response Cases - Case '1'.

Figure 98 is the *block diagram* that contains the code for transforming the *Memory Read (16-bit) Command*, case '2', into a readable format when it is received back from the microprocessor. The 32 bit CPU address is transposed from the 80186 format by the *Change 4([u8]) to [u32] VI*. The 16 bits of data that were requested by the user is converted from the 80186 format by the *Change 2([u8]) to [u16] VI* and then delineated from the other data by a leading equal sign. The command is read out to the user in an ASCII string format.

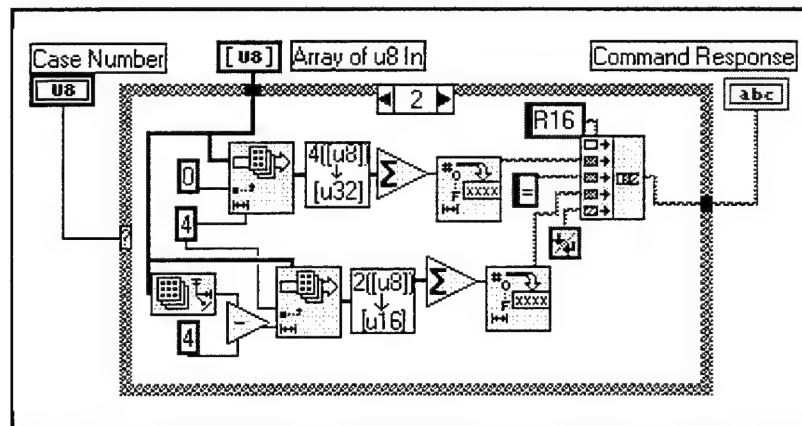


Figure 98: Block Diagram for VI Command Response Cases - Case '2'.

Figure 99 is the *block diagram* that contains the code for transforming the *Memory Write (16-bit) Command*, case '3', into a readable format when it is received back from the microprocessor. The 32 bit CPU address is transposed from the 80186 format by the *Change 4([u8]) to [u32] VI*. The 16 bits of data that were written to the memory are converted from the 80186 format by the *Change 2([u8]) to [u16] VI*. The command is repeated back to the user to confirm that the operation was carried out.

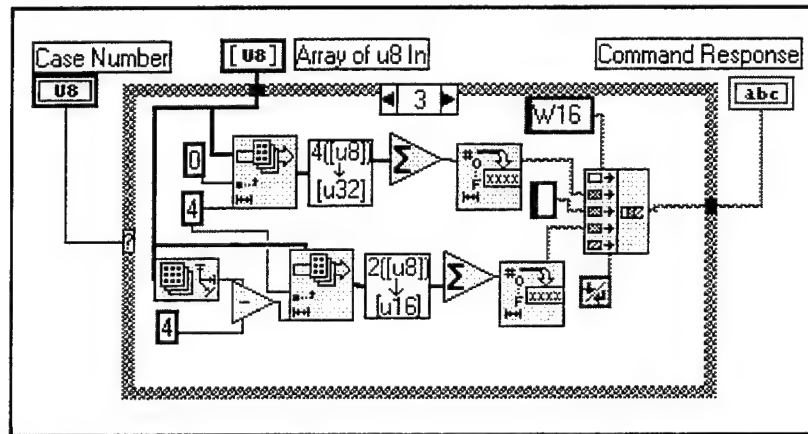


Figure 99: Block Diagram for VI Command Response Cases - Case '3'.

Figure 100 is the *block diagram* that contains the code for transforming the *I/O Port Read Command*, case '4', into a readable format when it is received back from the microprocessor. The 16 bit CPU I/O port number is transposed from the 80186 format by the *Change 2([u8]) to [u16] VI*. The 8-bits of data that were requested are delineated from the other data by a leading equal sign. The command is read out to the user in an ASCII string format.

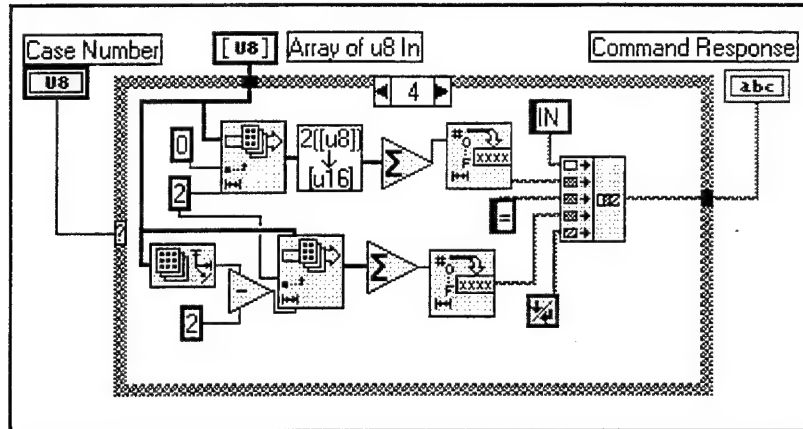


Figure 100: Block Diagram for VI Command Response Cases - Case '4'.

Figure 101 is the *block diagram* that contains the code for transforming the *I/O Port Write Command*, Case '5', into a readable format when it is received back from the microprocessor. The 16 bit CPU I/O port number is transposed from the 80186 format by the *Change 2([u8]) to [u16] VI*. The command is repeated back to the user in the same form in which it was sent confirming that the operation was carried out.

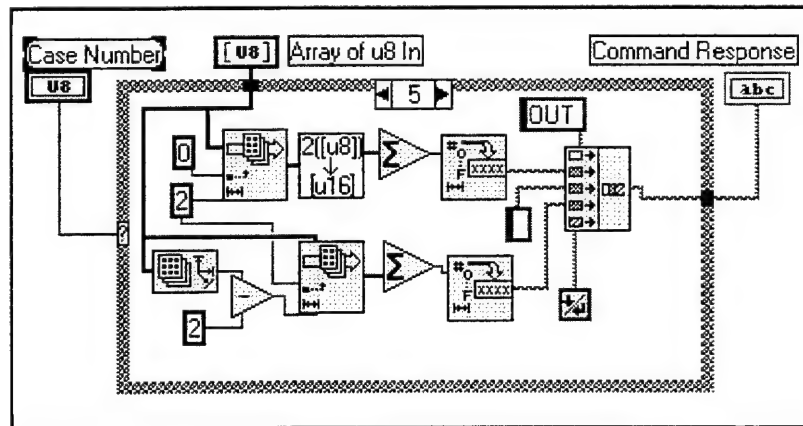


Figure 101: Block Diagram for VI Command Response Cases - Case '5'.

Figure 102 is the *block diagram* that contains the code for transforming the *PCB Read Command*, case '6', into a readable format when it is received back from the

Figure 104 is the *block diagram* that contains the code which corresponds to the *Run a File Command*, case '8'. This command is not repeated back to the user. Each individual command within the specified file is repeated back to the user. The code contained in case '8' serves as a place marker for future additions to the list of commands.

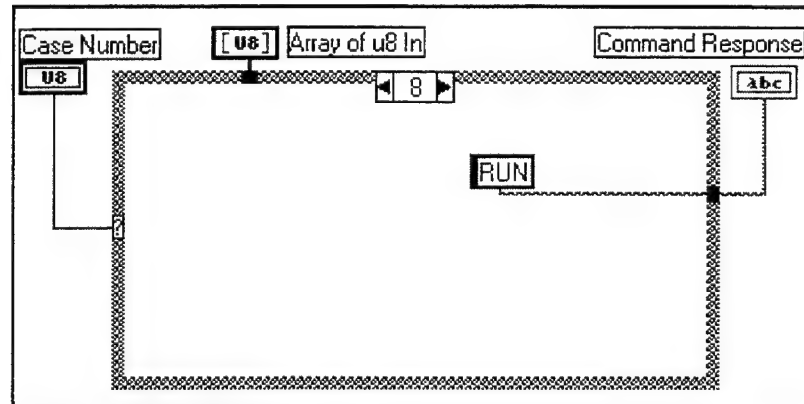


Figure 104: *Block Diagram for VI Command Response Cases - Case '8'.*

Figure 105 is the *block diagram* that contains the code which corresponds to the *Wait Command*, case '9'. This command is not sent to the microprocessor and is not repeated back to the user. This command is handled by the DCS interface software, by the *Wait VI*, within the LabVIEW environment using the PC's clock. The code contained in Case '9' serves as a place marker for future additions to the list of commands.

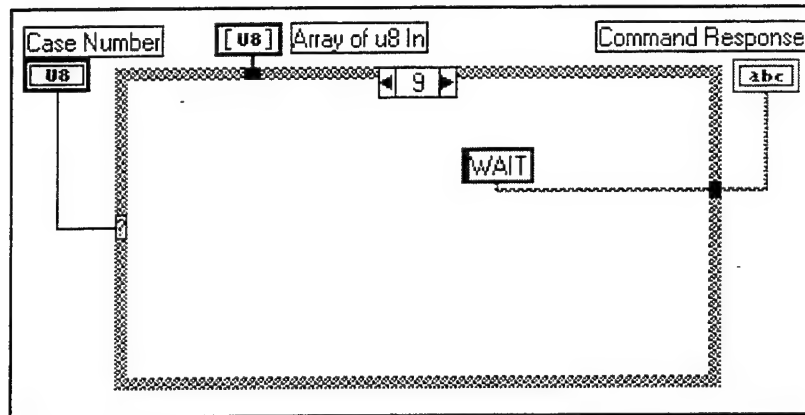


Figure 105: Block Diagram for VI Command Response Cases - Case '9'.

Figure 106 is the *block diagram* that contains the code which performs the *Enable Command*, Case '10'. This command is repeated back to the user in the same format in which it was sent, confirming that the embedded system is enabled.

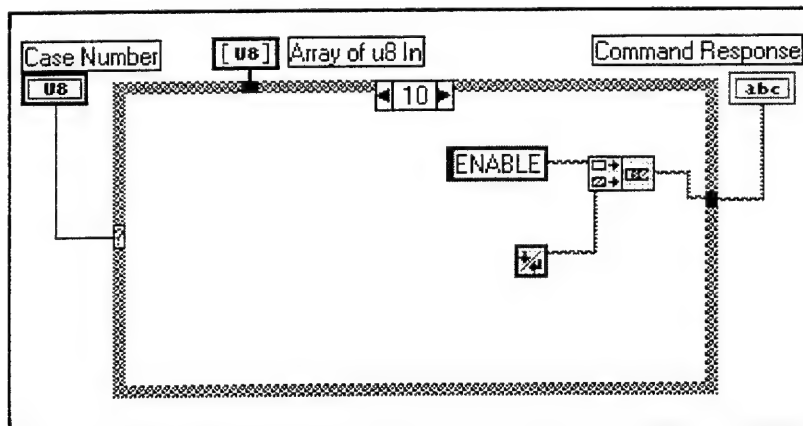


Figure 106: Block Diagram for VI Command Response Cases - Case '10'.

Figure 107 is the *block diagram* that contains the code which performs the *Disable Command*, Case '10'. This command is repeated back to the user in the same format in which it was sent, confirming that the embedded system is disabled.

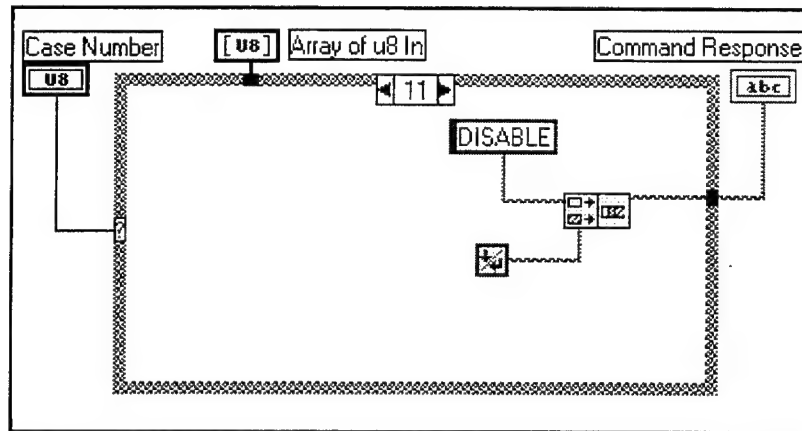


Figure 107: Block Diagram for VI Command Response Cases - Case '11'.

Figure 108 is the *block diagram* that contains the code which corresponds to the *Comment Command*, Case '12'. This command is not sent to the microprocessor and is not repeated back to the user. This command is handled by the DCS interface software, by the *Comment Handler VI*, within the LabVIEW environment. The code contained in Case '12' serves as a place marker for future additions to the list of commands.

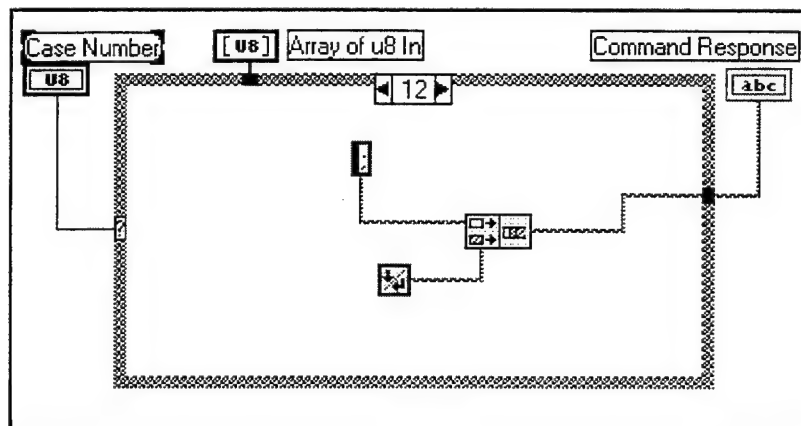


Figure 108: Block Diagram for VI Command Response Cases - Case '12'.

Figure 109 is the *block diagram* that contains the code for transforming the *Read A/D Converter Command*, case '13', into a readable format. At this time A/D commands are not handled by the System Control Board. A/D conversion takes place using the General Purpose Interface Bus (GPIB) using a multimeter and the *ADR Fake VI*. The *Read A/D Converter Command* is repeated back to the user in the same format that it was sent in with the addition of the measurement requested by the user. The requested measurement is set apart from the other data by a leading equal sign. Modifications to this case following the replacement of the Space Thermal Acoustic Refrigerator (STAR) System Control Board with the PANSAT System Control Board were addressed in Chapter IX.

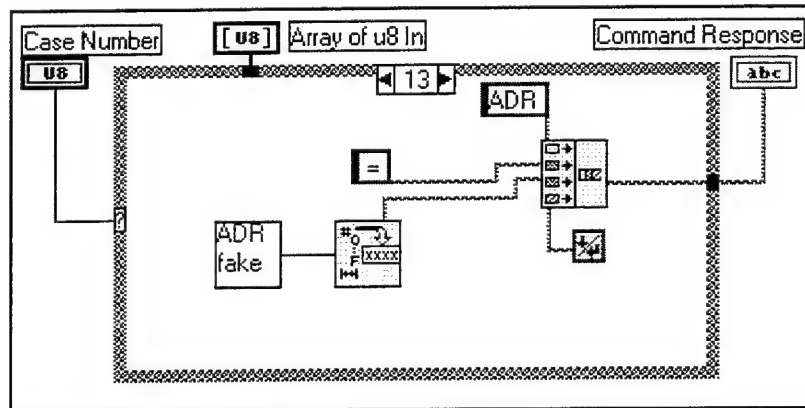


Figure 109: Block Diagram for VI Command Response Cases - Case '13'.

15. Send2 VI

Figure 110 shows the *icon and connectors* for VI *Send2*. This VI transmits the packet over the user specified serial port.

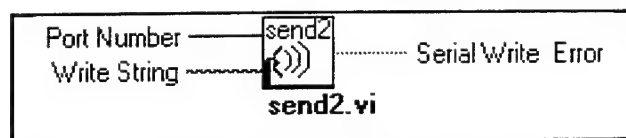


Figure 110: Icon and Connectors for VI *Send2*.

Figure 111 is the *block diagram* that performs the VI *Send2* function. VI *Send2* uses three other VIs previously discussed - the packet building *Send1 VI*, the *Serial Port Write VI* and the *Simple Error Handler VI*.

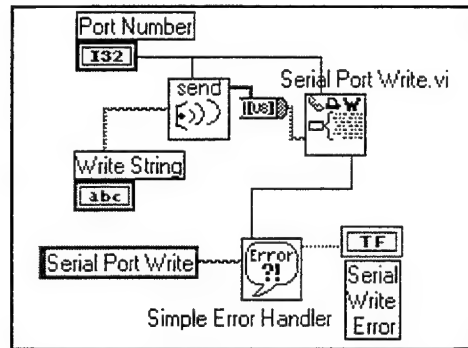


Figure 111: Block Diagram for VI *Send2*.

16. Command Packet VI

Figure 112 shows the *icon and connectors* for VI *Command Packet*. VI *Command Packet* determines whether a given command is valid and, if valid, assigns the command's corresponding hexadecimal number equivalent. The input command string is transformed into the 80186 format, making it ready for handling by the microprocessor.

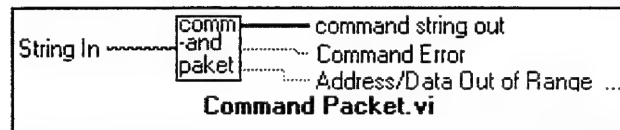


Figure 112: Icon and Connectors for VI *Command Packet*.

Figure 113 is the *block diagram* that contains the code for determining whether a given command is valid and, if valid, assigns the command's corresponding hexadecimal

number equivalent. VI *Command Packet* uses four previously described VIs - the *List of Commands VI*, the *Leading Space Stripper VI*, the *Change u16 to 2([u8]) VI* and the *Case Paket Select VI*.

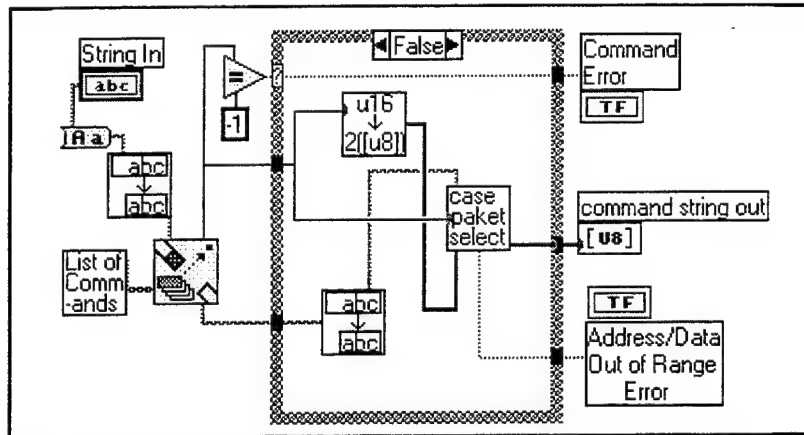


Figure 113: Block Diagram for VI Command Packet - 'False' Case.

Figure 114 shows the case where the input command does not match the list of commands. For this case, the 'true' case, an error is indicated to the user and operations cease. The correct form of the command can then be attempted again by the user.

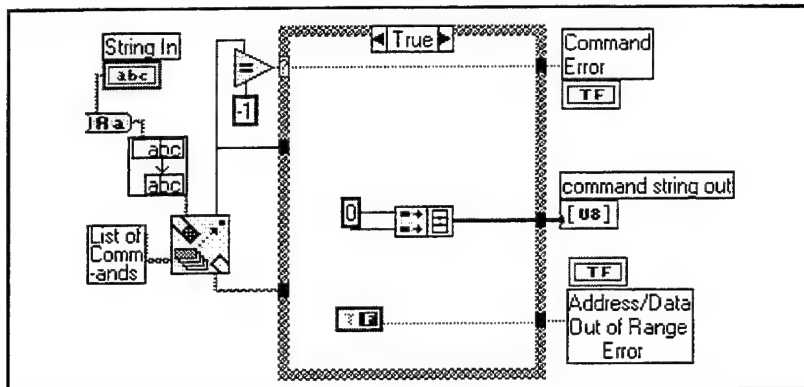


Figure 114: Block Diagram for VI Command Packet - 'True' Case.

17. Command Received

Figure 115 shows the *icon and connectors* for VI *Command Received*. This VI delineates the *data field* from the *Flag*, *Len1*, *Len2*, *CRC1* and *CRC2* bytes that are shown in Table 2 of Chapter IV.

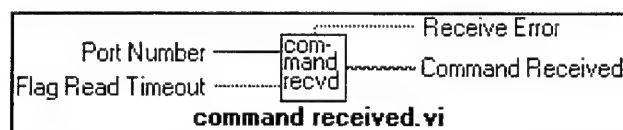


Figure 115: *Icon and Connectors for VI Command Received.*

Figure 116 is the *block diagram* that performs the VI *Command Received* function. VI *Command Received* uses the previously discussed VI *RCVR*. This VI indicates whether an error may have occurred when receiving data over the specified serial port.

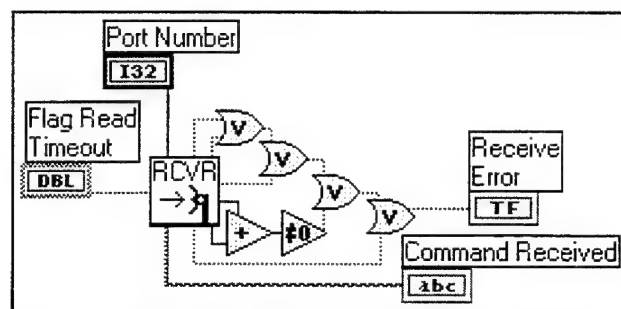


Figure 116: *Block Diagram for VI Command Received.*

18. Response Flag & Case Detect VI

Figure 117 shows the *icon and connectors* for VI *Response Flag & Case Detect*. This VI processes the *command data field* portion of the data sent from the microprocessor (see Chapter IV, Table 3). *Response Flag & Case Detect VI* checks to see if the *data field flag* is set and then checks to see that the hexadecimal number

corresponding to a command is within the *List of Commands* range. The *Array of U8 Out* is the *data field* less the *command data field*. The case number is determined and indicated.

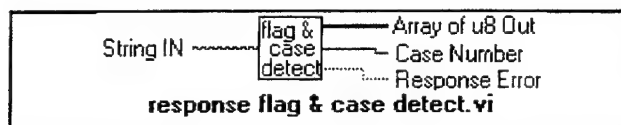


Figure 117: *Icon and Connectors for VI Response Flag & Case Detect.*

Figure 118 is the *block diagram* that performs the *VI Response Flag & Case Detect* function. *VI Response Flag & Case Detect* uses the previously discussed *VI Change 2([u8]) to [u16]*.

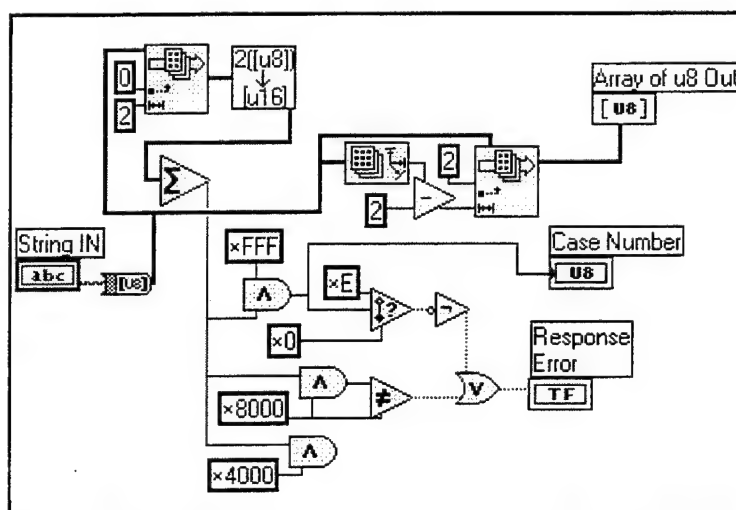


Figure 118: Block Diagram for VI Response Flag & Case Detect.

19. Determine Case VI

Figure 119 shows the *icon and connectors* for VI *Determine Case*. VI *Determine Case* processes the data field portion of data received from the microprocessor (see Table 2, Chapter IV).

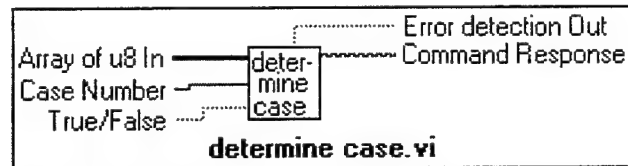


Figure 119: *Icon and Connectors for VI Determine Case.*

Figure 120 is the *block diagram* that performs the VI *Determine Case* function. VI *Determine Case* uses the previously discussed *Command Response Cases VI* to transform the information into a user friendly format.

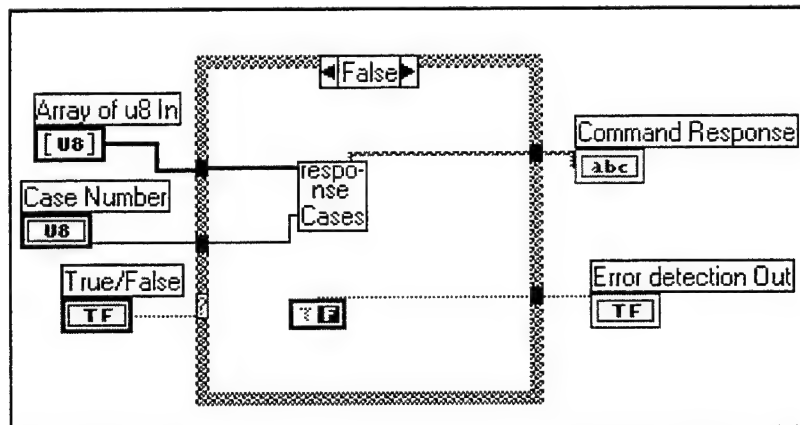


Figure 120: *Block Diagram for VI Determine Case - 'False' Case.*

Figure 121 shows the case in which a receiving error has been detected. If an error is detected, operations cease and the error is indicated to the user.

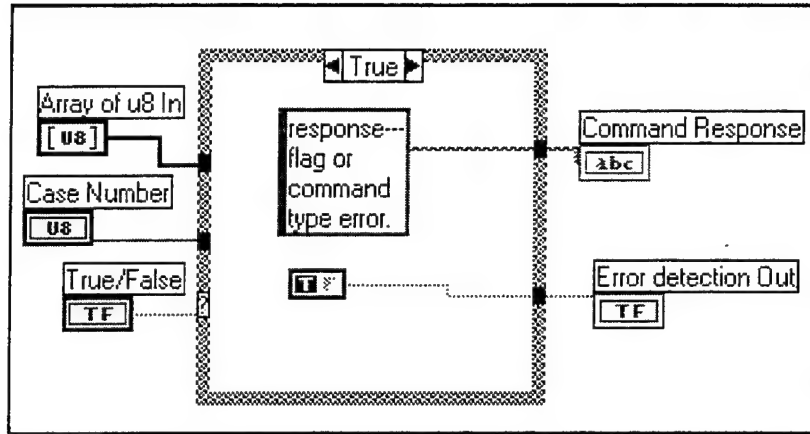


Figure 121: Block Diagram for VI Determine Case - 'True' Case.

20. Send Command VI

Figure 122 shows the *icon and connectors* for VI Send Command. *Send Command VI* is responsible for transforming the input command into an acceptable format for handling by the microprocessor. Once transformed, the command string is sent over the user specified serial port.

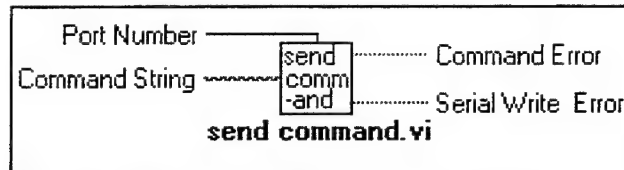


Figure 122: Icon and Connectors for VI Send Command.

Figure 123 shows the *block diagram* for VI Send Command (the 'false' case indicates that no errors were detected). *VI Send Command* uses two previously described VIs - *Command Packet VI* and *Send2 VI*.

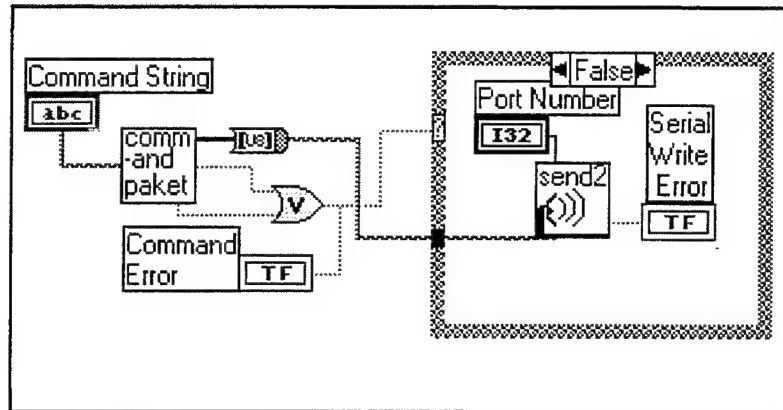


Figure 123: Block Diagram for VI Send Command - 'False' Case.

Figure 124 shows the 'true' case for VI *Send Command*. The 'true' case occurs when there has been a command error detected. When an error is detected, operations cease, the error is indicated and control returns back to the user.

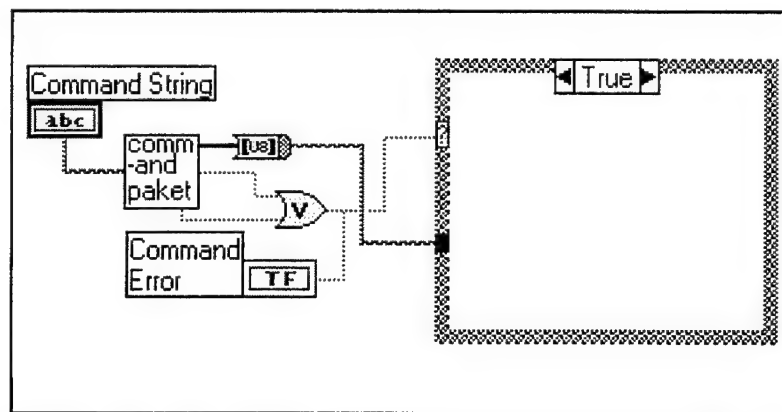


Figure 124: Block Diagram for VI Send Command - 'True' Case.

21. Command Response VI

Figure 125 shows the *icon and connectors* for *VI Command Response*. *Command Response VI* takes the *data field* portion of a *packet*, subtracts the *command data field* from it and transposes the remainder into a format that is legible by the user (see Tables 2 and 3 in Chapter IV).

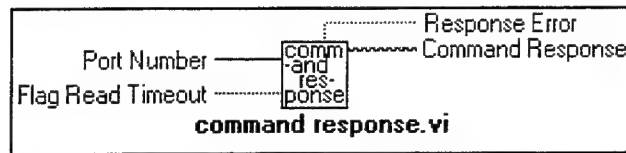


Figure 125: *Icon and Connectors for VI Command Response.*

Figure 126 is the *block diagram* or code used to perform the *Command Response VI's* function ('false' case indicates that no errors were previously detected). *Command Response VI* uses three previously discussed VIs - *Command Received VI*, *Response Flag & Case Detect VI* and *Determine Case VI*.

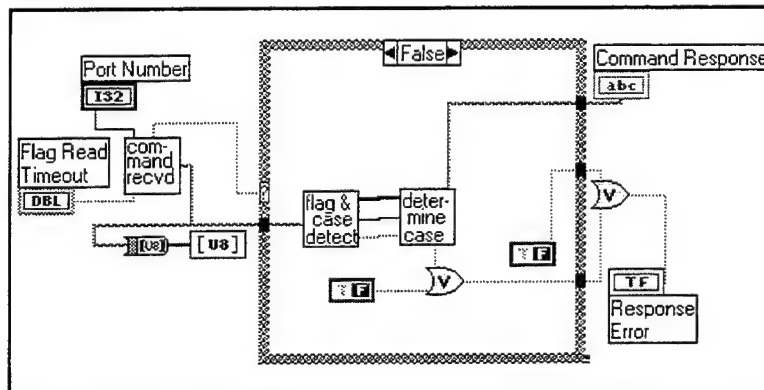


Figure 126: *Block Diagram for VI Command Response - 'False' Case.*

Figure 127 shows the 'true' case for *VI Command Response*. The 'true' case occurs when an error has been previously detected. If an error occurs, operations cease, the error is indicated and control returns to the user.

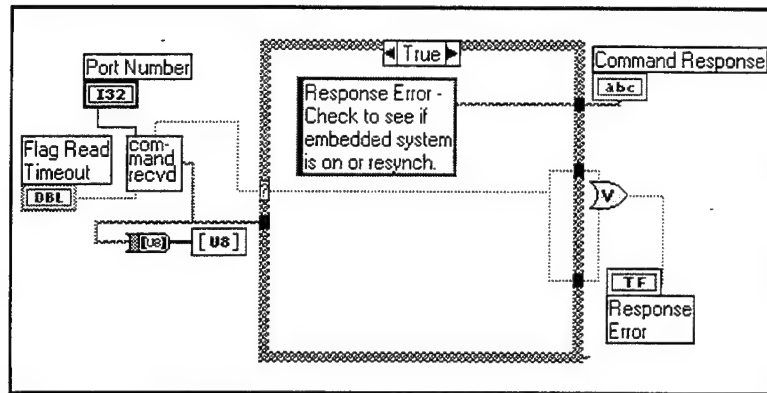


Figure 127: Block Diagram for VI Command Response - 'True' Case.

22. Find # of Commands VI

Figure 128 is the *icon and connectors* for VI *Find # of Commands*. *Find # of Commands VI* is a *subVI* of the *File Handler VI*, which is still to be discussed. *Find # of Commands VI* is used to determine the number of commands contained in a file.

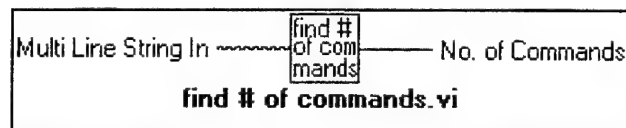


Figure 128: Icon and Connectors for VI *Find # of Commands*.

Figure 129 is the *block diagram* or code to carry out the function of VI *Find # of Commands*. *Find # of Commands VI* uses the previously discussed *Leading Space Stripper VI*.

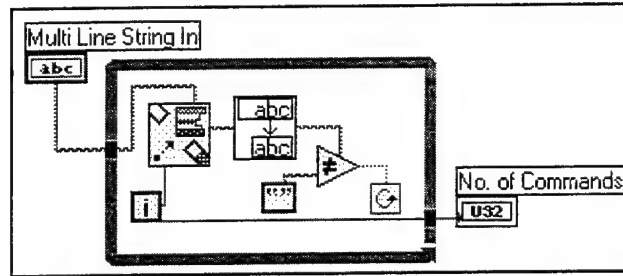


Figure 129: Block Diagram for VI Find # of Commands.

23. Case Number Find VI

Figure 130 is the *icon and connectors* for VI Case Number Find. Case Number Find VI is used to find a specified command. The command's hexadecimal number is equivalent to it's case number. The case number corresponds to it's position in the array of the previously discussed *List of Commands VI*.

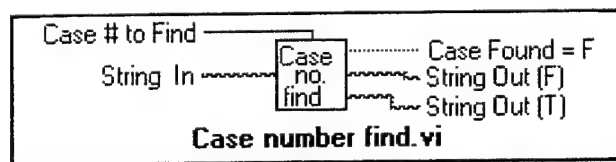


Figure 130: Icon and Connectors for VI Case Number Find.

Figure 131 is the *block diagram* or code for VI Case Number Find. If the command is found, a 'false' case is indicated on the *Case Found = F* output. If the command is not found a 'true' case is indicated. Two output strings are returned from the Case Number Find VI - *String Out (F)* and *String Out (T)*. *String Out (F)* is the string returned from the VI when the case is found. VI Case Number Find uses two previously discussed VIs - *Leading Space Stripper VI* and *List of Commands VI*.

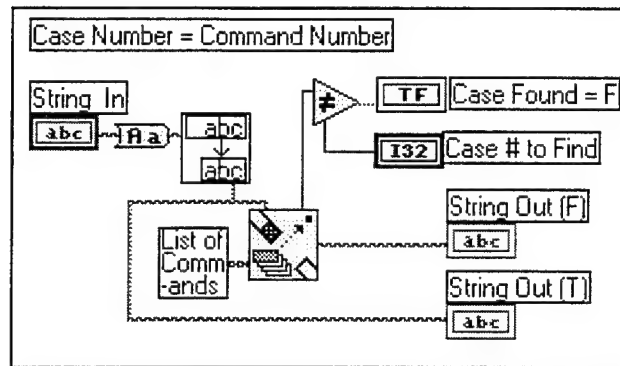


Figure 131: Block Diagram for VI Case Number Find.

24. Wait VI

Figure 132 is the *icon and connector* for the Wait VI. Wait VI is used to pause operations by the amount of time desired. Wait VI uses the PC clock and takes input in the millisecond scale.

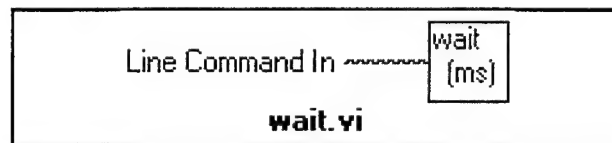


Figure 132: Icon and Connector for VI Wait.

Figure 133 is the *block diagram* or code for the VI Wait. VI Wait uses the previously discussed VI *Leading Space Stripper*.

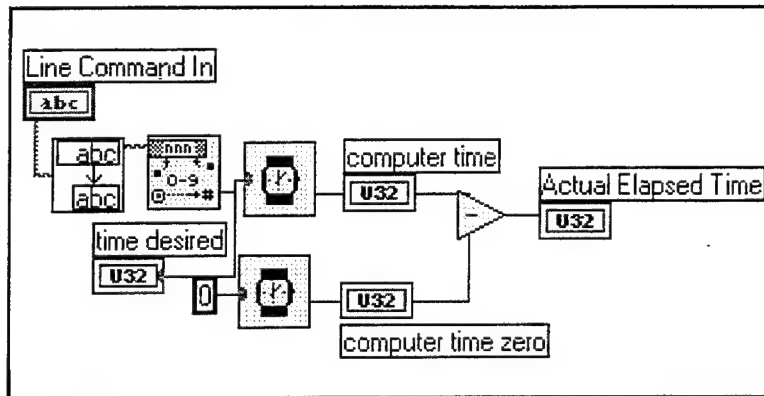


Figure 133: Block Diagram for VI Wait.

25. Data Out VI

Figure 134 is the *icon and correctors* for the *Data Out VI*. *Data Out VI* is used to set apart data that was retrieved from the embedded system.

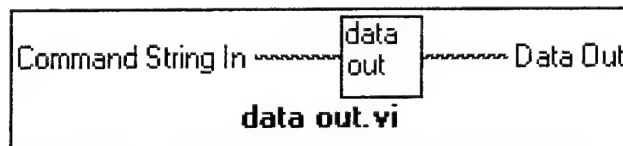


Figure 134: Icon and Connectors for VI Data Out.

Figure 135 is the *block diagram* for VI Data Out. The *Data Out VI* code takes each line received back from the microprocessor and searches for the equal sign. Once found, whatever data follows the equal sign is separated and indicated at the *Data Out connector*.

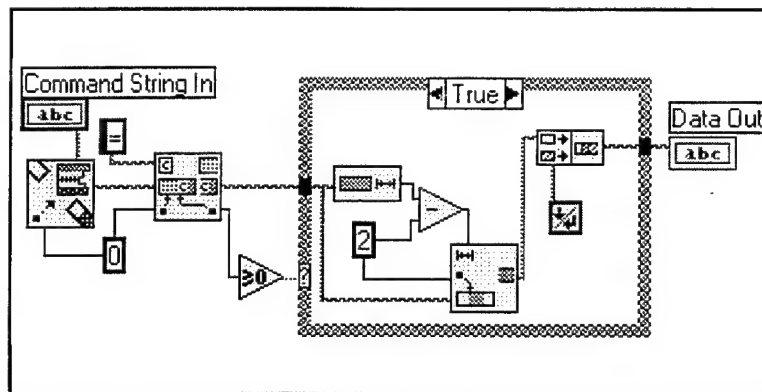


Figure 135: Block Diagram for VI Data Out - 'True' Case.

Figure 136 shows the 'false' case for VI Data Out. The 'false' case occurs when a line does not contain an equal sign.

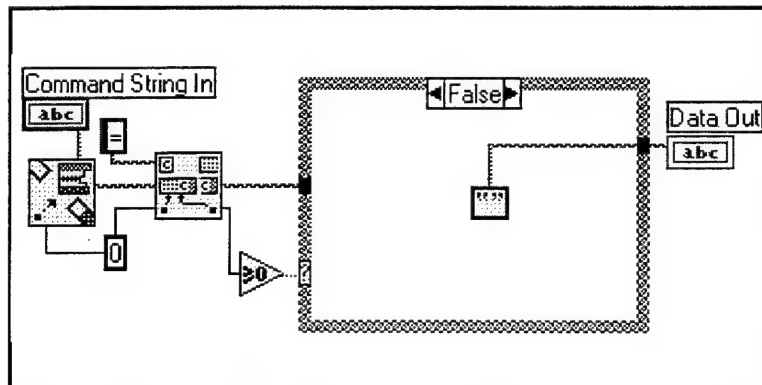


Figure 136: Block Diagram for VI Data Out - 'False' Case.

26. Control VI

Figure 137 is the *icon and connectors* for VI Control. VI Control is used to handle any of the eight low level commands plus the *Enable*, *Disable* and *A/D Read Commands*. These commands were discussed in Chapter VII in the Low and High Level Command Sections.

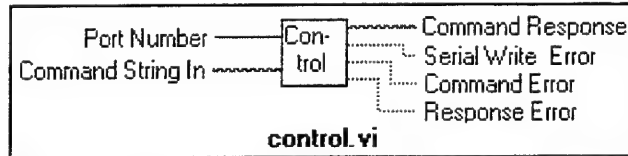


Figure 137: *Icon and Connectors for VI Control.*

Figure 138 is sequence '0' of the *block diagram* for VI Control. In this *block diagram* the command string is being formatted and sent to the microprocessor for handling. VI Control, sequence '0', uses the previously discussed VI Send Command.

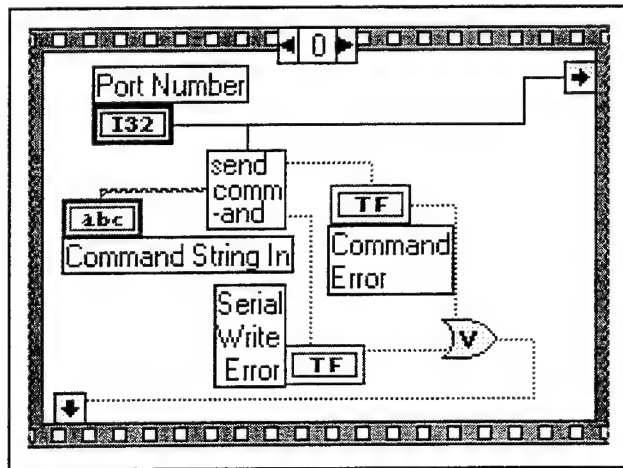


Figure 138: *Block Diagram for VI Control - Sequence '0'.*

Figure 139 is the 'false' case, sequence '1' of the *block diagram* for VI Control. In this *block diagram* the data string is received back from the microprocessor. The data string is transformed into a user legible format and indicated by the *Command Response* connector. VI Control, sequence '1', uses the previously discussed VI Command Response.

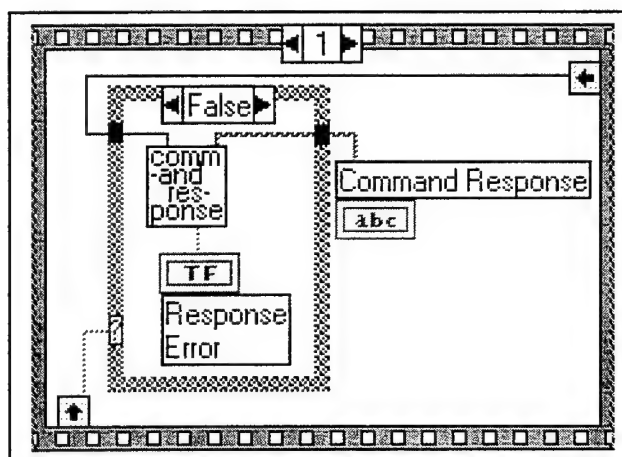


Figure 139: Block Diagram for VI Control - Sequence '1', 'False' Case.

Figure 140 is the 'true' case, sequence '1' of the *block diagram* for VI Control. The 'true' case occurs if an error is detected in the sending or receiving of data. If an error occurs operations cease, the error is indicated, and control is returned to the user.

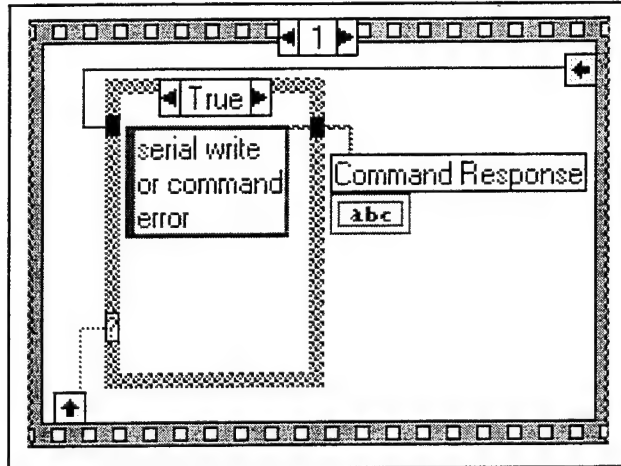


Figure 140: Block Diagram for VI Control - Sequence '1', 'True' Case.

27. Multi Line Control Rcvd Data to Array u16 VI

Figure 141 is the *icon and connector* for VI *Multi Line Control Rcvd Data to Array u16*. *Multi Line Control Rcvd Data to Array u16 VI* is usually used following the previously discussed *VI Multi Data Out*. Once data has been delineated using the equal sign, the data can then be stored into an array using the *Multi Line Control Rcvd Data to Array u16 VI*. This data can then be observed or manipulated further by the user.

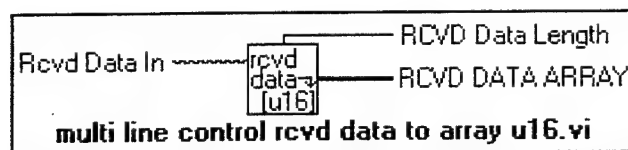


Figure 141: Icon and Connectors for VI *Multi Line Control Rcvd Data to Array u16*.

Figure 142 is the *block diagram* that performs the *VI Multi Line Control Rcvd Data to Array u16* function.

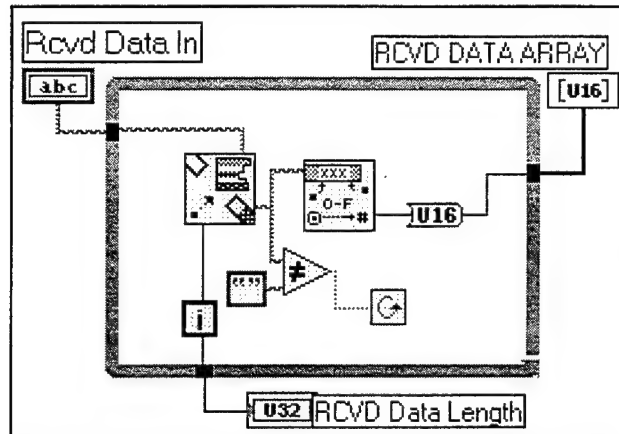


Figure 142: Block Diagram for VI Multi Line Control Rcvd Data to Array u16.

28. Comment Handler VI

Figure 143 is the *icon and connectors* for VI Comment Handler. *Comment Handler VI* enables the user to add comment lines into a file or sequence of commands. The semicolon is the symbol used for making comments.

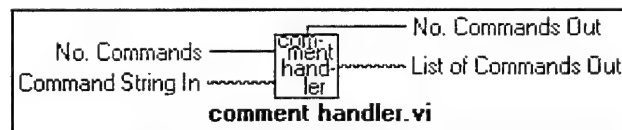


Figure 143: Icon and Connectors for VI Comment Handler.

Figure 144 is the *block diagram* for the *Comment Handler VI*. *Comment Handler VI* uses the previously discussed *Case Number Find VI*. Figure 144 is the 'true' case. The 'true' case occurs when a comment line (a line with a leading semicolon) is not found.

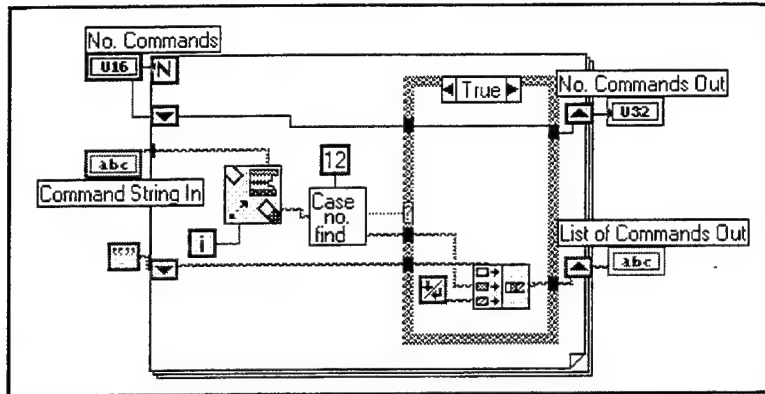


Figure 144: Block Diagram for VI Comment Handler - 'True' Case.

Figure 145 is the 'false' case block diagram for the *Comment Handler VI*. When a comment line (a line with a leading semicolon) is found, then this code is executed.

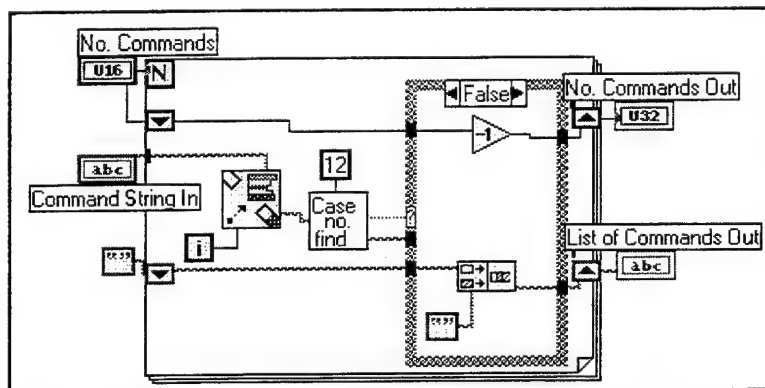


Figure 145: Block Diagram for VI Comment Handler - 'False' Case.

29. Multi Wait VI

Figure 146 is the *icon and connectors* for VI *Multi Wait*. *Multi Wait VI* performs the same function as the previously discussed *Wait VI*, but in a multiple command environment.

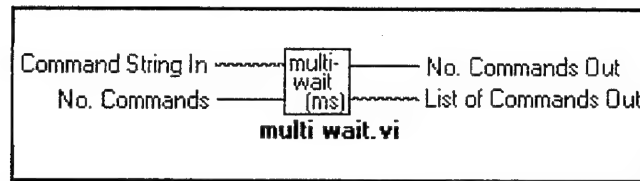


Figure 146: Icon and Connectors for VI Multi Wait.

Figure 147 is the 'false' case *block diagram* for VI Multi Wait. One line is processed at a time. The 'false' case is executed if the command line being processed is the *Wait Command* whose hexadecimal equivalent is '9'. *Wait Command VI* uses the previously discussed *Case Number Find VI* with '9' being the case sought.

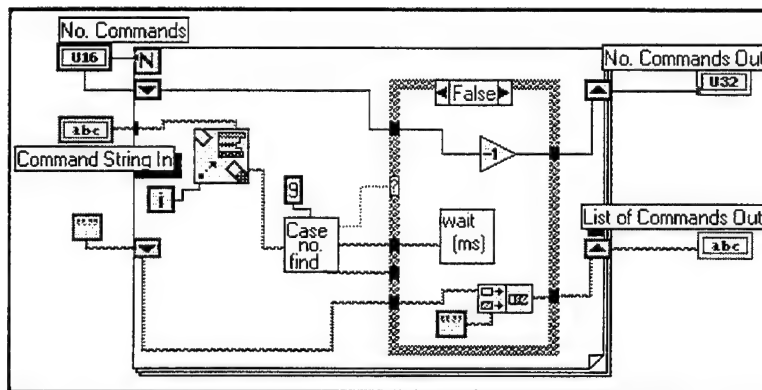


Figure 147: Block Diagram for VI Multi Wait - 'False' Case.

Figure 148 shows the 'true' case *block diagram* for VI Multi Wait. The 'true' case is executed if the *Wait Command* is not in the command line being processed. VI Multi Wait 'true' case uses the *Case Number Find VI*.

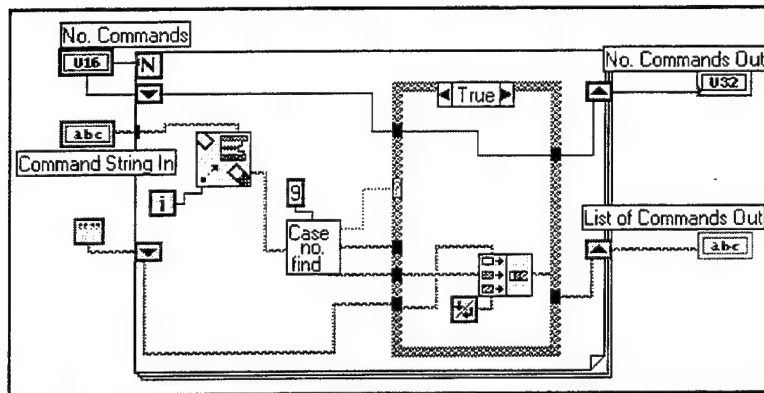


Figure 148: Block Diagram for VI Multi Wait - 'True' Case.

30. Multi-Line Data Out VI

Figure 149 is the *icon and connectors* for VI Multi-Line Data Out. Multi-Line Data Out VI collects the data delineated by the previously discussed VI Data Out.

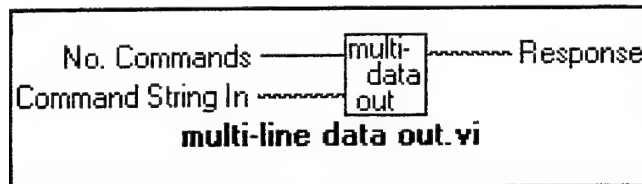


Figure 149: Icon and Connector for VI Multi-Line Data Out.

Figure 150 is the *block diagram* for VI Multi-Line Data Out. Multi-Line Data Out performs the same function as VI Data Out, but in a multiple command environment processing one command line at a time.

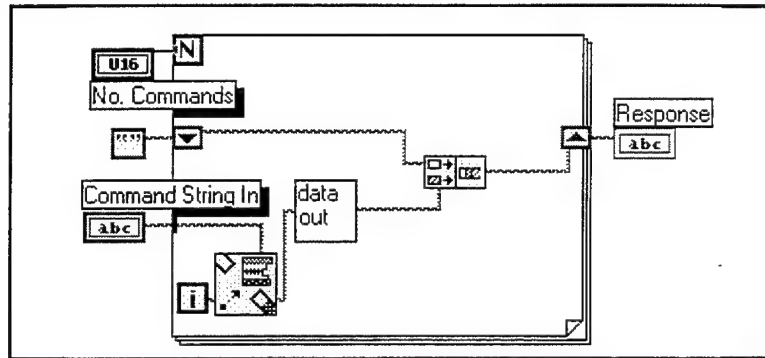


Figure 150: Block Diagram for VI Multi-Line Data Out.

31. Command Handler VI

Figure 151 is the *icon and connectors* for VI Command Handler. *Command Handler VI* performs the same function as the previously discussed *Control VI*, but handles multiple commands.

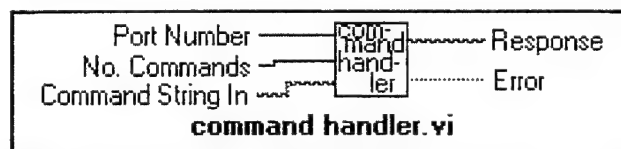


Figure 151: Icon and Connectors for VI Command Handler.

Figure 152 is the *block diagram* for VI Command Handler. *Command Handler VI* uses two previously discussed VIs - *Leading Space Stripper VI* and *Control VI*.

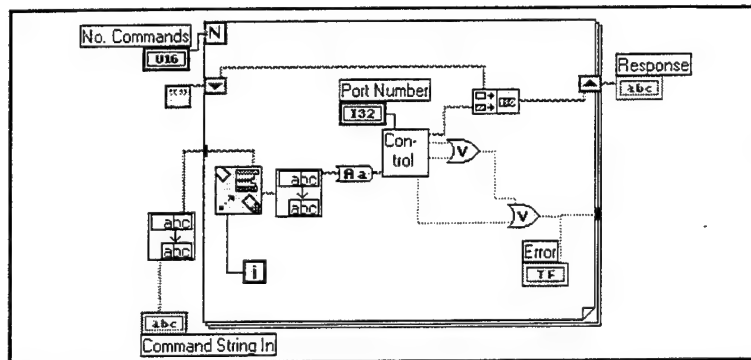


Figure 152: Block Diagram for VI Command Handler.

32. File Handler VI

Figure 153 is the *icon and connectors* for the *File Handler VI*. *File Handler VI* is used to open files created using the *Window's* application *Notepad*.

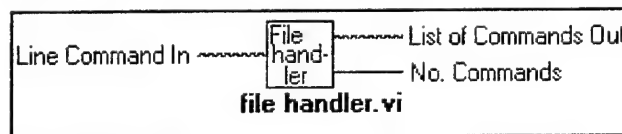


Figure 153: Icon and Connectors for VI File Handler.

Figure 154 is the 'false' case *block diagram* for VI *File Handler*. *File Handler VI* uses three previously discussed VIs - *Case Number Find VI*, *Leading Space Stripper VI* and *Find # of Commands VI*.

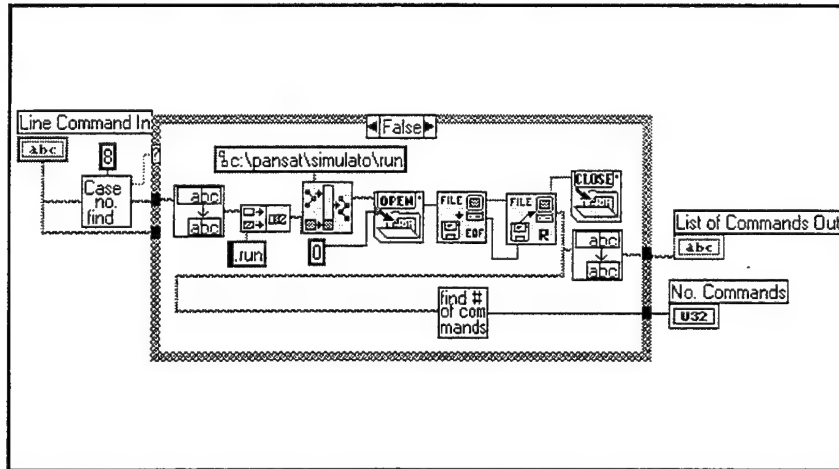


Figure 154: Block Diagram for VI File Handler - 'False' Case.

Figure 155 is the 'true' case block diagram for VI File Handler. The 'true' case is executed if case number '8' is not found.

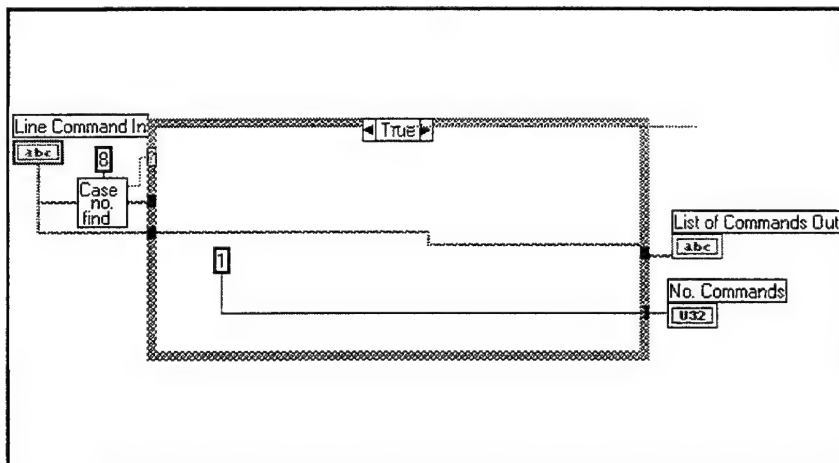


Figure 155: Block Diagram for VI File Handler - 'True' Case.

33. Multi Line Control VI

Figure 156 is the icon and connectors for the Multi Line Control VI. Multi Line Control VI handles files, waits, comments, data outputs, and stores the data outputs into

an array. Additionally, *Multi Line Control VI* handles all the same commands as the *Control VI*, but in a multiple command environment.

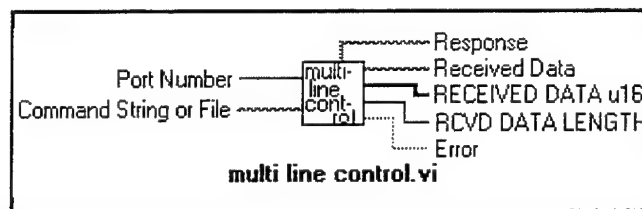


Figure 156: Icon and Connectors for VI Multi Line Control.

Figure 157 is the 'true' case *block diagram* for VI Multi Line Control. *Multi Line Control VI* 'true' case *block diagram* uses seven previously discussed VIs - *Leading Space Stripper VI*, *File Handler VI*, *Multi Wait VI*, *Comment Handler VI*, *Command Handler VI*, *Multi-Line Data Out VI* and *Multi Line Control Rcvd Data to Array u16 VI*.

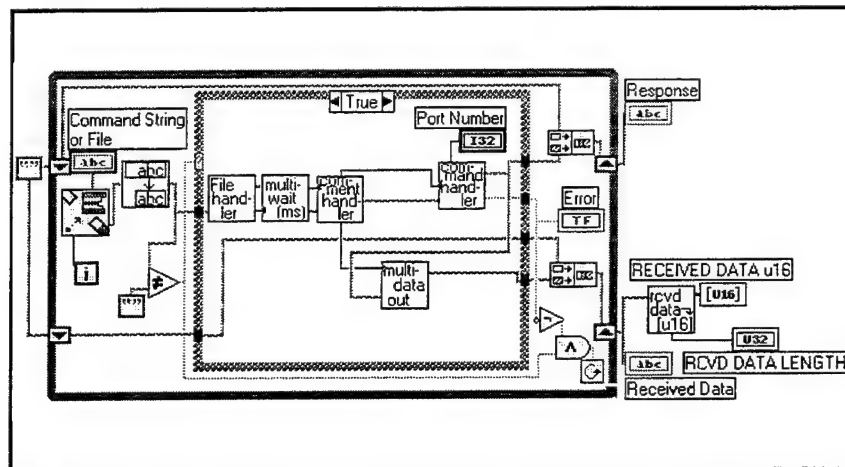


Figure 157: Block Diagram for VI Multi Line Control - 'True' Case.

Figure 158 is the 'false' case *block diagram* for VI Multi Line Control. The 'false' case is executed when all input commands have been executed.

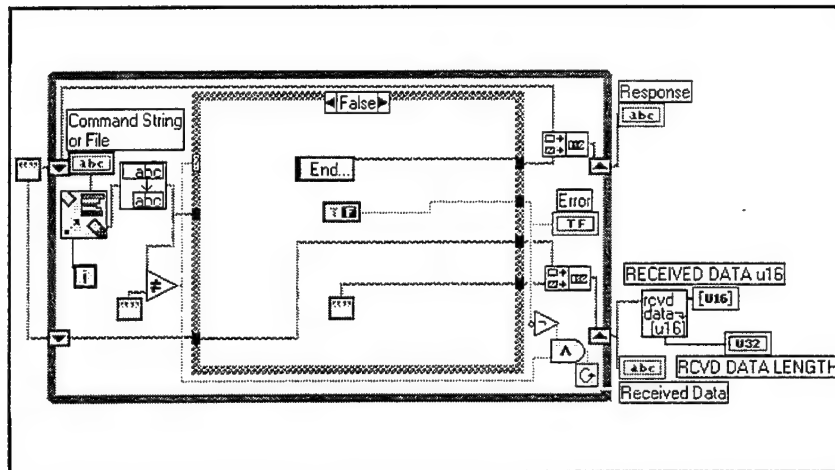


Figure 158: Block Diagram for VI Multi Line Control - 'False' Case.

B. TSWEEP3 VI

TSWEEP3 VI is a modified version of the *TSWEEP2 VI* that was briefly discussed in Chapter VII. *TSWEEP3 VI* uses the DCS Interface Software discussed in Section A to test the temperature multiplexer. *TSWEEP2 VI* was used to test 16 temperature sensors whereas *TSWEEP3 VI* has been developed to test a variable number of temperature sensors. Figure 159 is the *icon and connectors* for VI *TSWEEP3*.

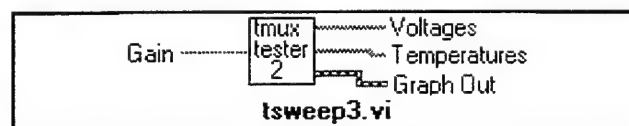


Figure 159: Icon and Connectors for VI *TSWEEP3*.

Figure 160 is the *block diagram* for VI *TSWEEP3*. This *block diagram* has a couple of minor changes compared to the *block diagram* of *TSWEEP2 VI*. The first change is that the command line on the left hand side has the additional *run tmux2 command* and the second change is that the *Convert to Celsius VI* has been replaced with the *Convert to Celsius2 VI*. Testing is ongoing on the TMUX, as stated in Chapter VII, and now the second set of 16 sensors are being tested along with the first 16. This required the changes to *TSWEEP2 VI* just discussed. The new file, *tmux2*, was easy to create. The *tmux1* file was copied and then saved as *tmux2*. Corresponding addresses for the second set of sensors were then applied to the new file.

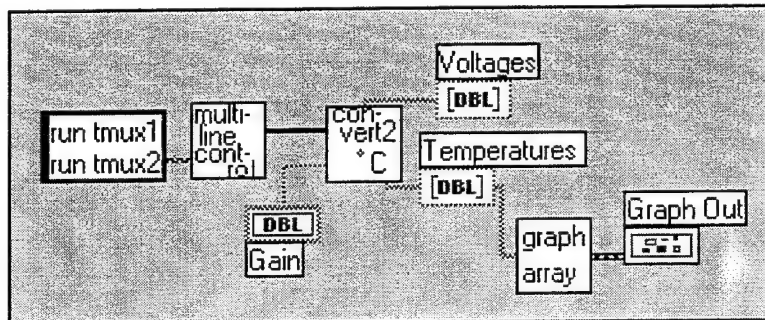


Figure 160: *Block Diagram for VI TSWEEP3.*

Multi Line Control VI is a *subVI* of the *TSWEEP3 VI*. *Multi Line Control VI* was discussed in Section A of this Appendix and therefore is not discussed here. *Convert to Celsius2 VI* and *Graph Array VI* and their *subVIs* are discussed below.

1. Convert to Celsius2 VI

Figure 161 is the *icon and connectors* for VI *Convert to Celsius2*. *Convert to Celsius2 VI* is a modified version of *Convert to Celsius VI*. The changes allow for the testing of any number of temperature sensors, whereas before testing was limited to the

16 sensors on one channel. This need was identified during the testing of the second channel of the TMUX, which led to these modifications.

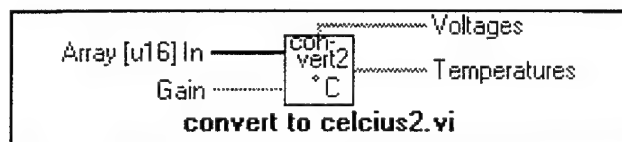


Figure 161: Icon and Connectors for VI *Convert to Celsius2*.

Figure 162 is the *block diagram* for VI *Convert to Celsius2*. *Meas. to Volt VI* and *Temp. Const. List VI* are *subVIs* of the *Convert to Celsius2 VI* and are discussed below. The outer *For Loop* shown in Figure 162 is executed according to the number of sensor measurements taken. An algorithm is then carried out on each measurement which converts the voltage into a temperature.

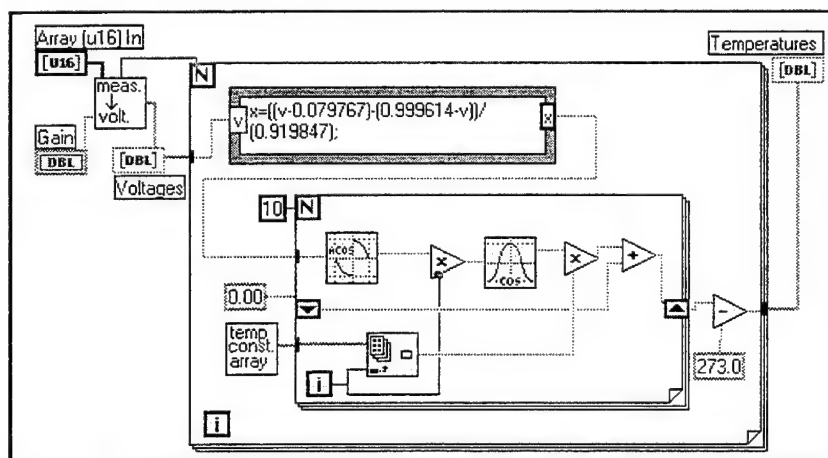


Figure 162: Block Diagram for VI *Convert to Celsius2*.

2. Meas. to Volt VI

Figure 163 is the *icon and connectors* for VI *Meas. to Volt*. *Meas. to Volt VI* takes the temperature sensor voltages, prepares them for further processing and indicates the number of sensor readings that were made.

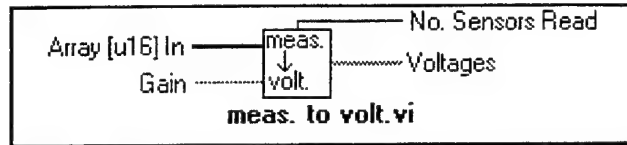


Figure 163: *Icon and Connectors for VI Meas. to Volt.*

Figure 164 is the *block diagram* for the *Meas. to Volt VI*. Each element of the unsigned 16 bit number array, shown in Figure 164 labeled as *Array [u16] In*, is converted into a double precision floating point number, processed and then stored in a double precision floating point number array labeled *Voltages*.

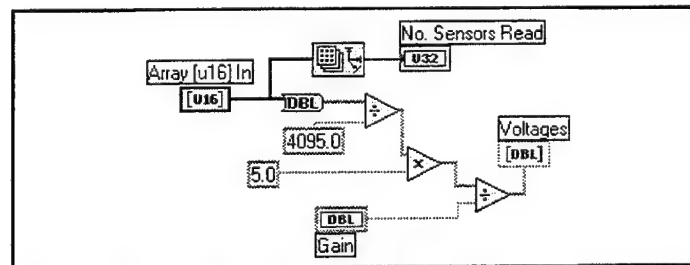


Figure 164: *Block Diagram for VI Meas. to Volt.*

3. Temp. Const. List VI

Figure 165 is the *icon and connectors* for the *Temp. Const. List VI*. *Temp. Const. List VI* contains a list of constants used in the algorithm to convert voltage into temperature readings.

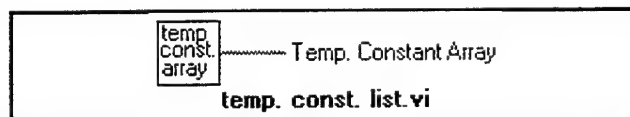


Figure 165: *Icon and Connectors for VI Temp. Const. List.*

Figure 166 is the *block diagram* for the *Temp. Const. List VI*. VI *Temp. Const. List* is an 11 element array, containing real numbers with seven digits of significance.

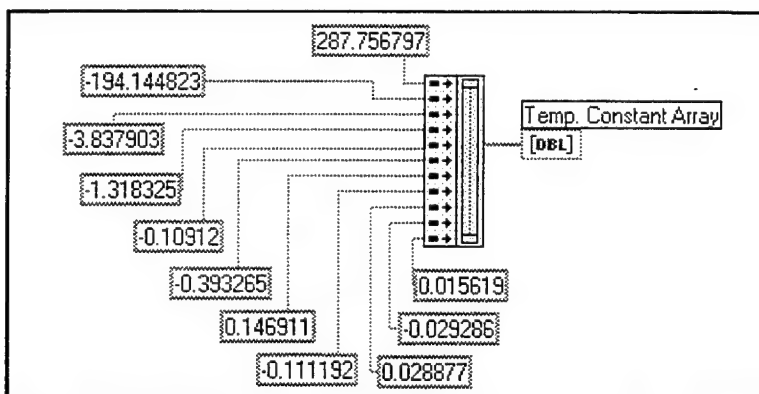


Figure 166: Block Diagram for VI *Temp. Const. List*.

4. Graph Array VI

Figure 167 is the *icon and connectors* for the *Graph Array VI*. *Graph Array VI* is *subVI* of *TSWEEP3 VI* used to plot the temperatures following the voltage conversions.

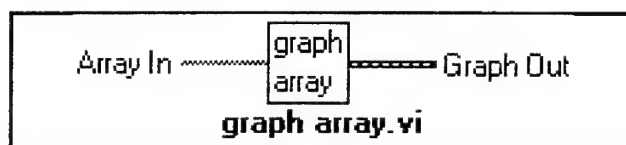


Figure 167: Icon and Connectors for VI *Graph Array*.

Figure 168 is the *block diagram* for the *Graph Array VI*. The previously determined temperature measurements are indicated by the *Array In* input shown in Figure 168. These values are then plotted out.

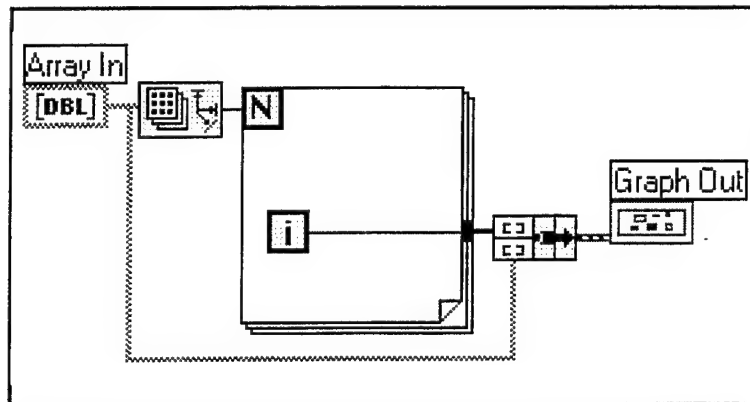


Figure 168: *Block Diagram for VI Graph Array.*

APPENDIX B. SIMULATOR ELECTRICAL POWER SUBSYSTEM (EPS) INTERFACE SOFTWARE CODE

The software described in Appendix B is presented in the same order as it was in Chapter IV. Each virtual instrument (VI) in this appendix is presented with its *icon and connectors* and *block diagrams*. Development of the VIs in this appendix are ongoing, waiting the installation and arrival of the prototype batteries. *EPS Battery Telemetry VI* and *EPS Roll Rate/Attitude VI block diagrams* contain only initial thoughts that require further development.

A. POWER SUPPLY DRIVER

Figure 169 contains the *icon and connectors* for VI *GPIBTEST*. *GPIBTEST VI* controls the operation of HP6653A Power Supply using remote programming and the General Purpose Interface Bus (GPIB) [Ref. 11: pp. 6-1 through 7-27].

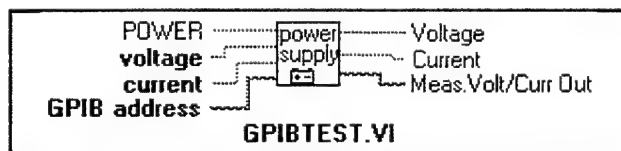


Figure 169: *Icon and Connectors for VI GPIBTEST.*

Figure 170 is the 'false' case *block diagram* for *GPIBTEST VI*. The 'false' case occurs when the power switch to the VI is turned off.

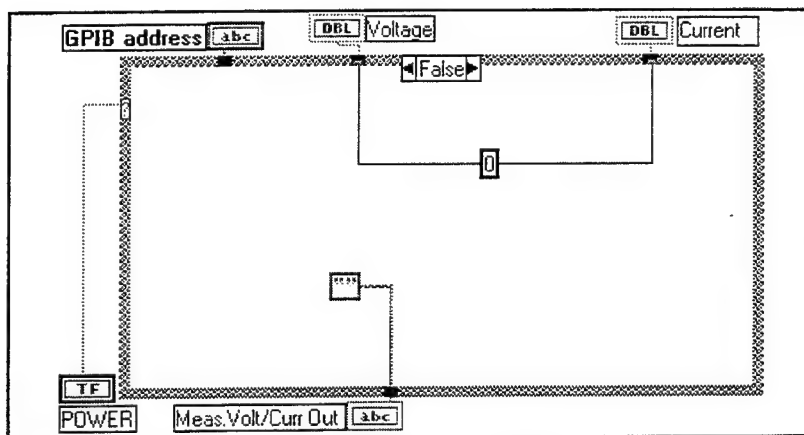


Figure 170: Block Diagram for VI GPIBTEST - 'False' Case.

Figure 171 is the 'true' case, sequence '0' for GPIBTEST VI. In this sequence remote commands are sent to the Power Supply directing it to take the specified voltage and current. Remote programming commands are found in the Power Supply's operating manual [Ref. 13: pp. 6-1 through 6-9].

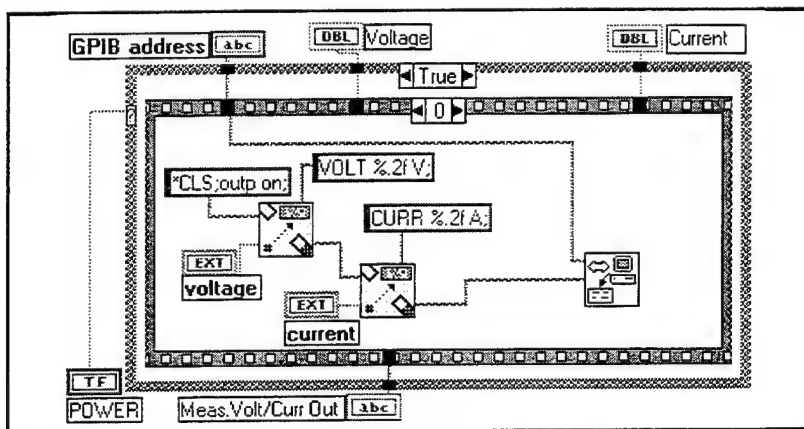


Figure 171: Block Diagram for VI GPIBTEST - Sequence '0', 'True' Case.

Figure 172 is the 'true' case, sequence '1' for *GPIBTEST VI*. In this sequence, operations are paused allowing the Power Supply time to set it's output voltage and current to the specified level.

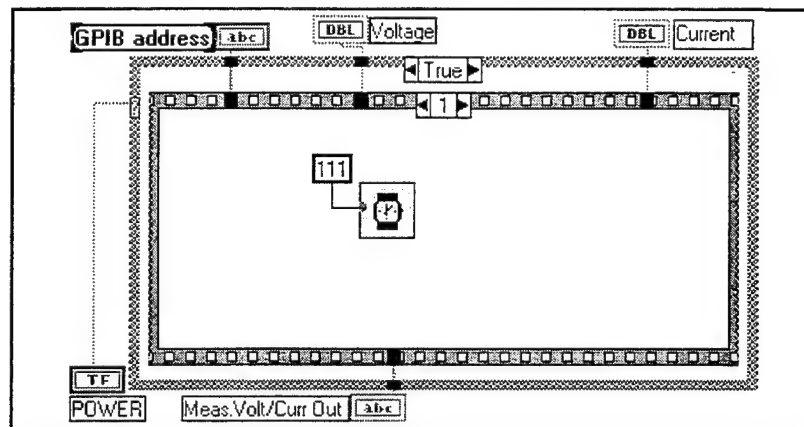


Figure 172: Block Diagram for VI *GPIBTEST* - Sequence '1', 'True' Case.

Figure 173 is the 'true' case, sequence '2' for *GPIBTEST VI*. In this sequence, voltage and current measurements of the Power Supply are requested.

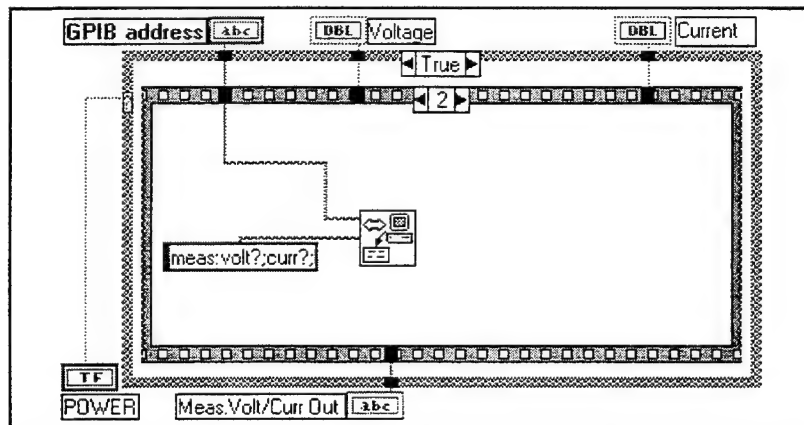


Figure 173: Block Diagram for VI *GPIBTEST* - Sequence '2', 'True' Case.

Figure 174 is the 'true' case, sequence '3' for *GPIBTEST VI*. In this sequence, the voltage and current measurements are read from the Power Supply and then formatted for easy reading to the user.

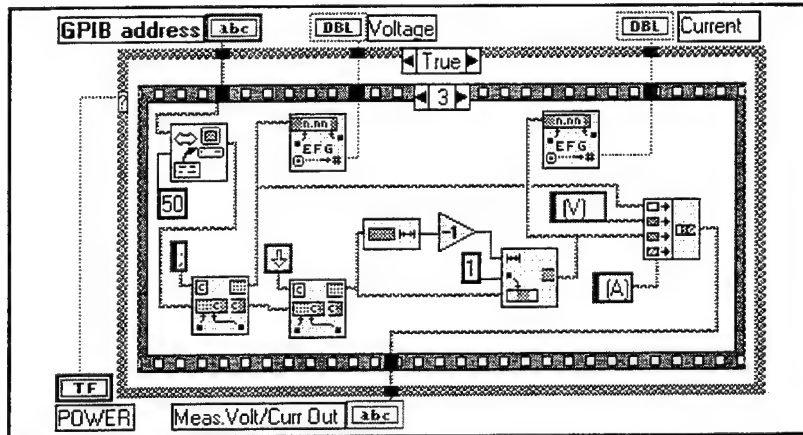


Figure 174: Block Diagram for VI *GPIBTEST* - Sequence '3', 'True' Case.

B. GPIBLOAD VI

Figure 175 is the *icon and connectors* for VI *GPIBLOAD*. *GPIBLOAD VI* is used to control the HP6060A Single Input Electronic Load. *GPIBLOAD VI* has three modes of operation - constant current (CC), constant resistance (CR) and constant voltage. The CC mode is the one that the simulator is expected to operate in, therefore the CC mode is discussed below.

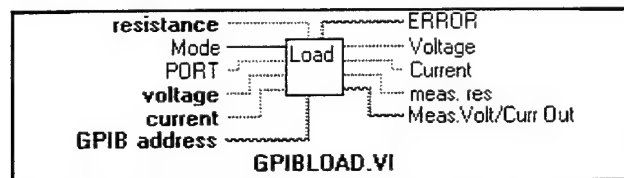


Figure 175: Icon and Connectors for VI *GPIBLOAD*.

Figure 176 is sequence '0', 'false' and 'CC' case (mode) *block diagram* for VI *GPIBLOAD*. To the left of Figure 176 is the 'false' case for the PORT0 remote command. This command is used to control the general-purpose digital port on the rear of the Electronic Load. The Port output becomes active when the PORTON ('true' case) command is used and becomes inactive when the PORTOFF ('false' case) command is used [Ref. 14: p. 2-17]. *GPIBLOAD VI* is defaulted to the 'true' case. Sequence '0' sets the voltage and current inputs on the HP6060A Single Input Load using remote programming commands [Ref. 18: pp. 1-1 through 5-10].

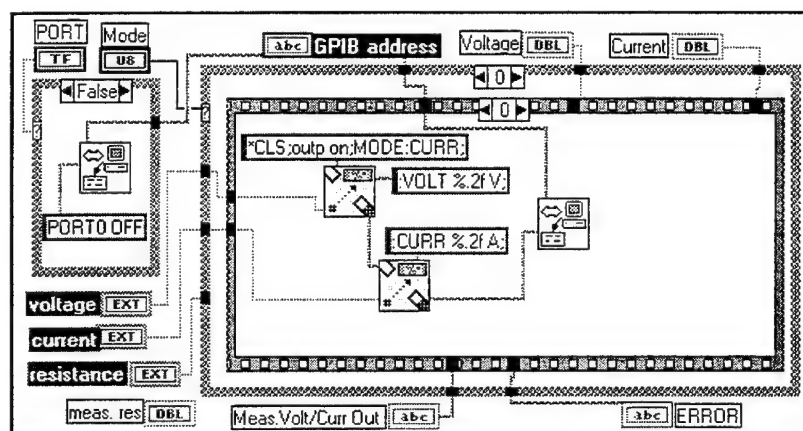


Figure 176: *Block Diagram for VI GPIBLOAD - Sequence '0', 'False' and 'CC' Case.*

Figure 177 is sequence '1', 'false' and 'CC' case *block diagram* for VI *GPIBLOAD*. Sequence '1' pauses operations allowing the Load time to register the input voltage and current settings.

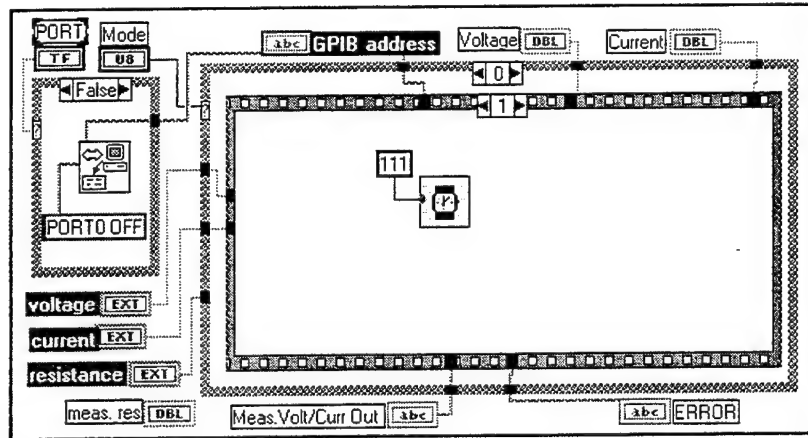


Figure 177: Block Diagram for VI GPIBLOAD - Sequence '1', 'False' and 'CC' Case.

Figure 178 is sequence '2', 'true' and 'CC' case *block diagram* for VI GPIBLOAD. Sequence '2' queries the Load to check and see if any errors have occurred in the input.

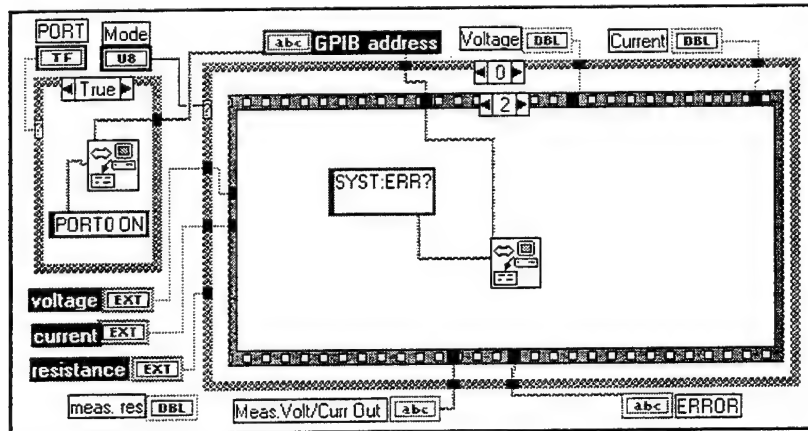


Figure 178: Block Diagram for VI GPIBLOAD - Sequence '2', 'True' and 'CC' Case.

Figure 179 is sequence '3', 'true' and 'CC' case *block diagram* for VI GPIBLOAD. Sequence '3' reads the Load indicating to the user any error messages that have occurred.

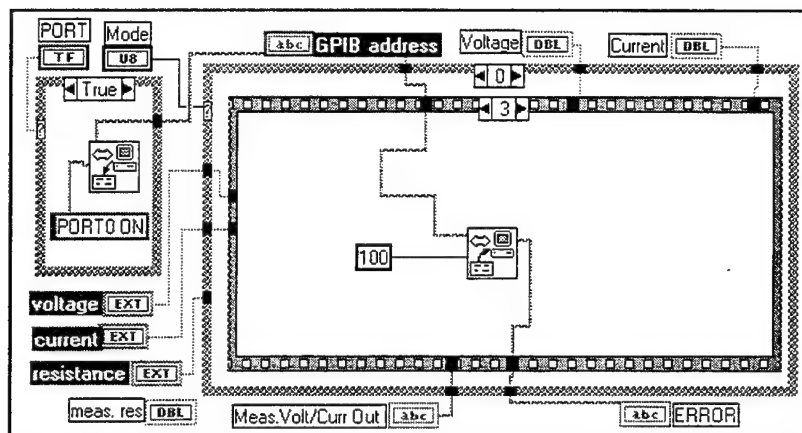


Figure 179: Block Diagram for VI GPIBLOAD - Sequence '3', 'True' and 'CC' Case.

Figure 180 is sequence '4', 'true' and 'CC' case *block diagram* for VI GPIBLOAD. Sequence '4' queries the Electronic Load for output current and voltage measurements.

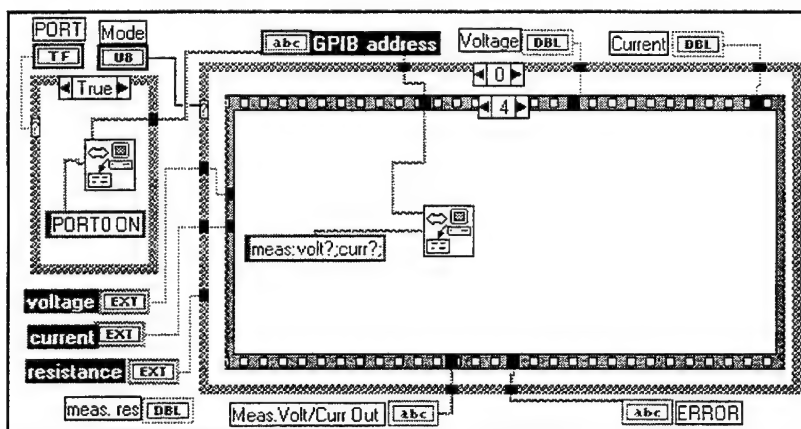


Figure 180: Block Diagram for VI GPIBLOAD - Sequence '4', 'True' and 'CC' Case.

Figure 181 is sequence '5', 'true' and 'CC' case *block diagram* for VI GPIBLOAD. Sequence '5' reads the Electronic Load for output current and voltage measurements and then formats it for easy reading by the user.

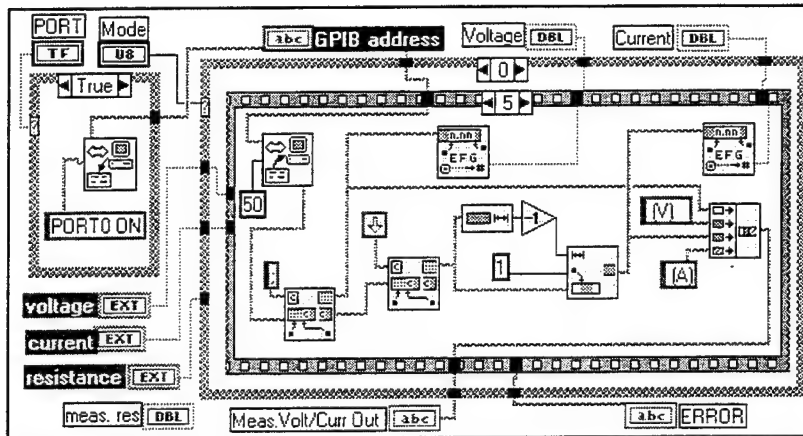


Figure 181: Block Diagram for VI GPIBLOAD - Sequence '5', 'True' and 'CC' Case.

C. SUPLOAD VI

Figure 182 is the *icon and connectors* for VI SUPLOAD. SUPLOAD VI is used to test the operation of the Power Supply connected to the Electronic Load.

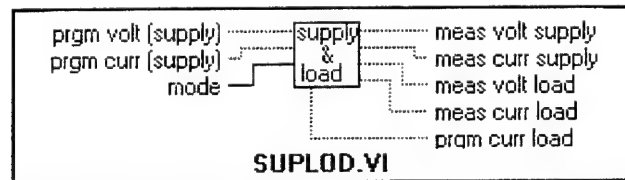


Figure 182: Icon and Connectors for VI SUPLOAD.

Figure 183 is sequence '0' of the *block diagram* for VI SUPLOAD. Sequence '0' uses the GPIBLOAD VI. An input current is applied to the Load. Measured voltage and current are then indicated.

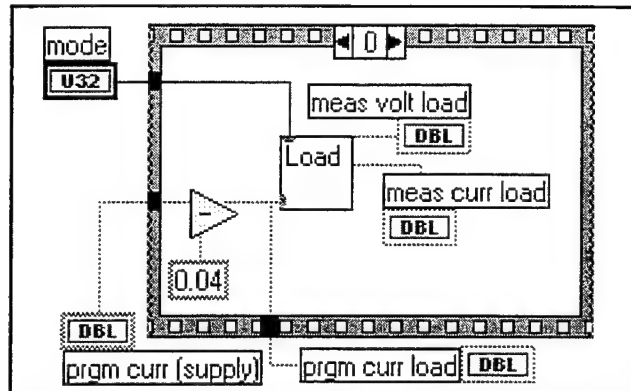


Figure 183: Block Diagram for VI SUPLOAD - Sequence '0'.

Figure 184 is sequence '1' of the *block diagram* for VI SUPLOAD. Sequence '1' pauses operations for SUPLOAD VI to allow the Electronic Load Device to stabilize.

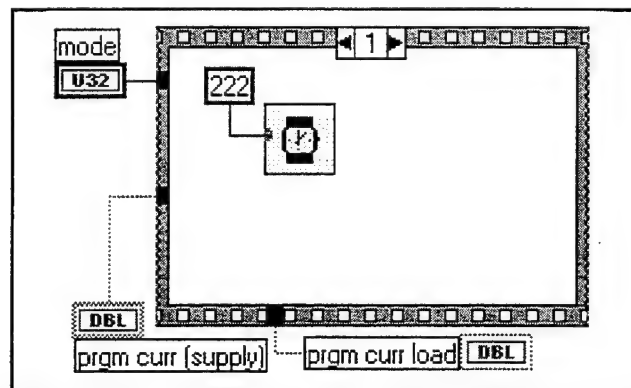


Figure 184: Block Diagram for VI SUPLOAD - Sequence '1'.

Figure 185 is sequence '2' of the *block diagram* for VI SUPLOAD. Sequence '2' uses the GPIBTEST VI the driver for the Power Supply. An input current and voltage is applied to the Power Supply. Measured voltage and current are then indicated.

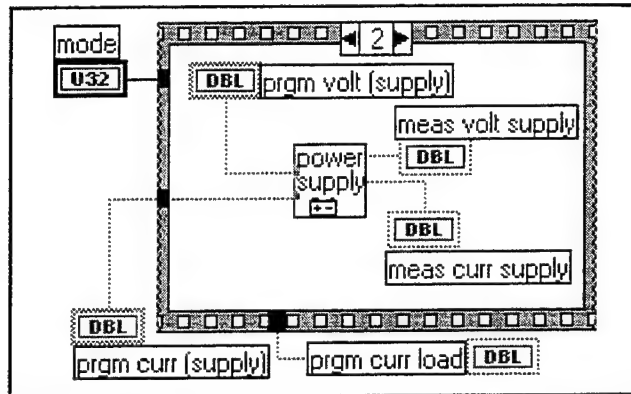


Figure 185: Block Diagram for VI SUPLOAD - Sequence '2'.

D. EPS BATTERY TELEMETRY VI

Figure 186 is the *icon and connectors* for VI EPS Battery Telemetry. EPS Battery Telemetry VI will be used to provide the user a means for checking the status of the batteries. Development of this VI is ongoing, awaiting the arrival and installation of the prototype batteries.

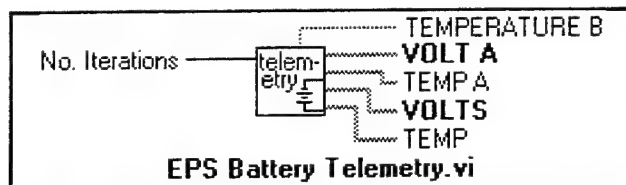


Figure 186: Icon and Connectors for VI EPS Battery Telemetry.

Figure 187 is the *block diagram* for VI EPS Battery Telemetry. EPS Battery Telemetry VI uses three previously discussed VIs - Multi Line Control VI (with its numerous VIs), Convert To Celsius2 VI and Graph Array Out VI.

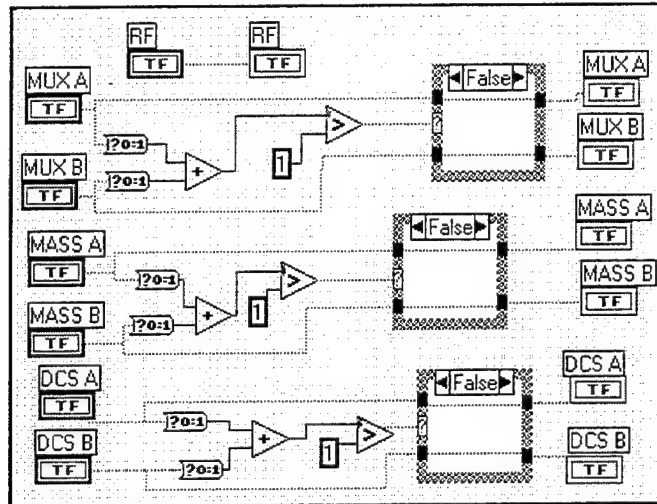


Figure 188: Block Diagram for VI EPS Subsystem On/Off - 'False' Case.

Figure 189 is the 'true' case *block diagram* for VI EPS Subsystem On/Off. A 'true' case indicates that both of the 'true' case subsystems are being switched on. The block diagram shows that if someone attempts to turn on any paired subsystem concurrently, a default mechanism is invoked, turning subsystem A on and subsystem B off.

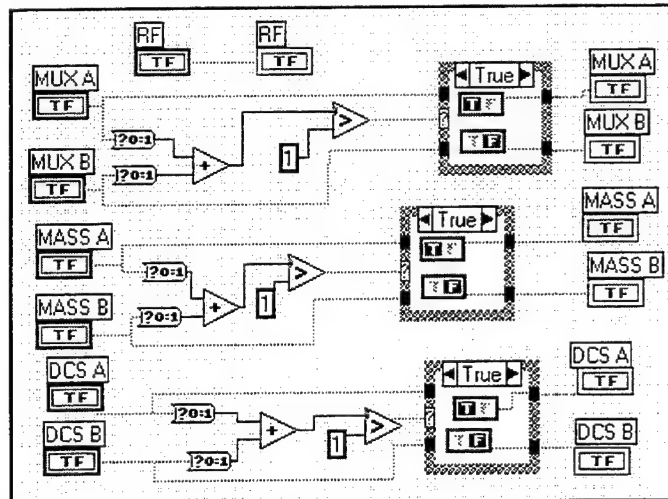


Figure 189: Block Diagram for VI EPS Subsystem On/Off - 'True' Case.

F. EPS ROLL RATE/ATTITUDE VI

Figure 190 is the *icon and connectors* for VI EPS Roll Rate/Attitude. EPS Roll Rate/Attitude VI will be used to provide the user a means to determine the satellite attitude and roll rate. This is achieved by strategically distributing eight current sensors on the satellite solar panels. Using the power supply to simulate the solar panels, further testing and development of this platform can continue.

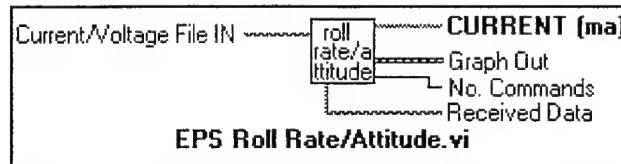


Figure 190: Icon and Connectors for VI EPS Roll Rate/Attitude.

Figure 191 is the *block diagram* for VI EPS Roll Rate/Attitude. Further development of this idea is necessary.

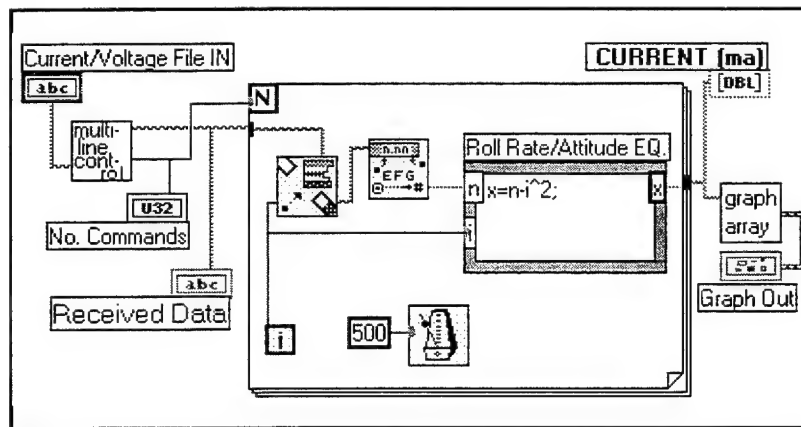


Figure 191: Block Diagram for VI EPS Roll Rate/Attitude.

G. IV FILE SOLAR SIMULATOR VI

Figure 192 is the *icon and connectors* for VI IV File Solar Simulator VI. IV File Solar Simulator VI has been developed to allow the testing of the prototype batteries

during a user specified number of simulated orbit. The orbit is simulated using a current/voltage (IV) input file.

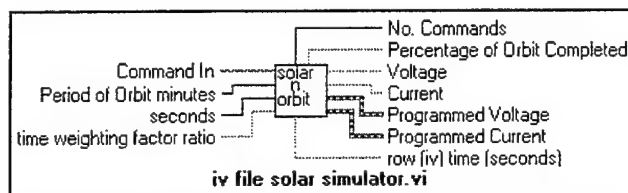


Figure 192: Icon and Connectors for VI IV File Solar Simulator.

Figure 193 is the *block diagram* for VI IV File Solar Simulator. The file that contains the input current and voltage values is fed into the power supply. A time weighing factor allows the speeding up or slowing down of real time processing. The orbit period is adjustable. Percent of orbit completion is graphically displayed to the user along with output voltage and current measurements.

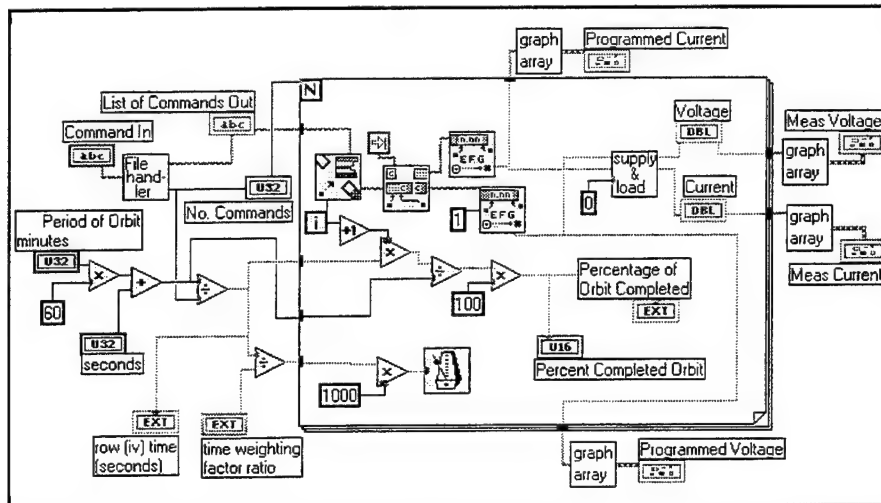


Figure 193: Block Diagram for VI IV File Solar Simulator.

APPENDIX C. PERIPHERAL CONTROL BUS (PCB) ADDRESSES CURRENTLY DEFINED.

Table 9 is a listing of subsystem Peripheral Control Bus (PCB) addresses currently defined.

System Name	System Address (Binary)	System Address (Hexadecimal)
RF Subsystem	0000, 0001	0, 1
Electrical Power Subsystem	1000, 1001	8, 9
System Control A	0010, 1011	2, 3
System Control B	1010, 1011	A, B
Analog MUX A	0100, 0101	4, 5
Analog MUX B	1100, 1101	C, D
Mass Storage A	0110, 0111	6, 7
Mass Storage B	1110, 1111	E, F

Table 9: PCB Subsystem Addresses [Ref. 7: p. 7].

LIST OF REFERENCES

1. Space Systems Academic Group, *PANSAT Engineering Design Review (EDR)*, Naval Postgraduate School, Monterey, CA, March 1994.
2. Morris, L. T., *Petite Amateur Navy Satellite - A Proof of Concept, Half-Duplex, Digital Spread-Spectrum, Store-and-Forward Communication Satellite*, Space Systems Academic Group, Naval Postgraduate School, Monterey, CA, May 1994.
3. Horning, J. A., "Navy Education Through Amateur Radio Satellite Development," Proc. of the AIAA Space Programs and Technologies Conf., Huntsville, AL, September 1993.
4. Sakoda, D. J., "Naval Postgraduate School Spread Spectrum Communication Satellite," Proc. of the AIAA Space Programs and Technologies Conf., Huntsville, AL, September 1993.
5. Price, H., *BAX Reference Manual*, Bekttek Corporation, Bethel Park, PA, March 1992.
6. Sakoda, D. J., Horning, J. A. and Rigmaiden, D. W., *PANSAT Digital Control Subsystem (DCS) Hardware Design Description*, Space Systems Academic Group, Naval Postgraduate School, Monterey, CA, March 1994.
7. Horning, J. A., *PANSAT Software Detailed Design Document*, Space Systems Academic Group, Naval Postgraduate School, Monterey, CA, March 1994.
8. Phelps, R. L., *PANSAT Electrical Power Subsystem Hardware Design Description*, Space Systems Academic Group, Naval Postgraduate School, Monterey, CA, July 1994.
9. National Instruments Corporation, *LabVIEW - GPIB and Serial Port VI Reference Manual*, Austin, TX, August 1993 Edition.
10. National Instruments Corporation, *Getting Started with Your AT-GPIB and the NI-488.2 Software for MS-DOS*, Austin, TX, January 1993.
11. National Instruments Corporation, *Using Your NI-488.2 Software with Microsoft Windows*, Austin, TX, April 1993.

12. National Instruments Corporation, *LabVIEW Basics Course Manual*, Austin, TX, January 1994.
13. Hewlett-Packard Company, *Operating Manual HP-IB DC Power Supplies Series 665xA*, July 1990.
14. Hewlett-Packard Company, *Operating Manual 300 Watt Single Input Electronic Load HP Model 6060A*, December 1988.
15. National Instruments Corporation, *LabVIEW for Windows - Tutorial*, Austin, TX, August 1993 Edition.
16. National Instruments Corporation, *LabVIEW for Windows -User Manual*, Austin, TX, August 1993 Edition.
17. Hewlett-Packard Company, *HP3478A Multimeter Operator's Manual*, Palo Alto CA, January 1988.
18. Hewlett-Packard Company, *Programming Reference Guide Hewlett-Packard Electronic Load Family*, December 1988.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
Cameron Station
Alexandria, Virginia 22304-6145
2. Library, Code 52 2
Naval Postgraduate School
Monterey, California 93943-5101
3. Chairman, Code EC 1
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, California 93943-5121
4. Chairman, Code SP 1
Space Systems Academic Group
Naval Postgraduate School
Monterey, California 93943-5110
5. Prof. Rudy Panholzer, Code SP 1
Space Systems Academic Group
Naval Postgraduate School
Monterey, California 93943-5110
6. Prof. Tri Ha, Code EC 1
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, California 93943-5121
7. J. A. Horning, Code SP 1
Space Systems Academic Group
Naval Postgraduate School
Monterey, California 93943-5110
8. Captain Thompson 1
Office of the Chief of Naval Operations
Code N63, Room 4E679, The Pentagon
Washington, DC 20350-2000

-
9. Commander, Naval Space Command 1
ATTN: N112
5280 4th Street
Dahlgren, VA 2248-5300
10. LT T. C. Calvert 2
1268 W. 5100 S.
Riverdale, Utah 84405